Einführung in Visual Computing

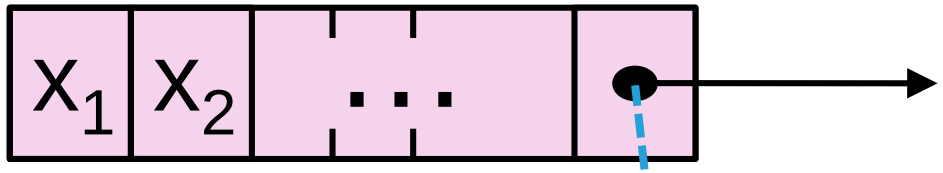
186.822

Polygon Filling

Werner Purgathofer

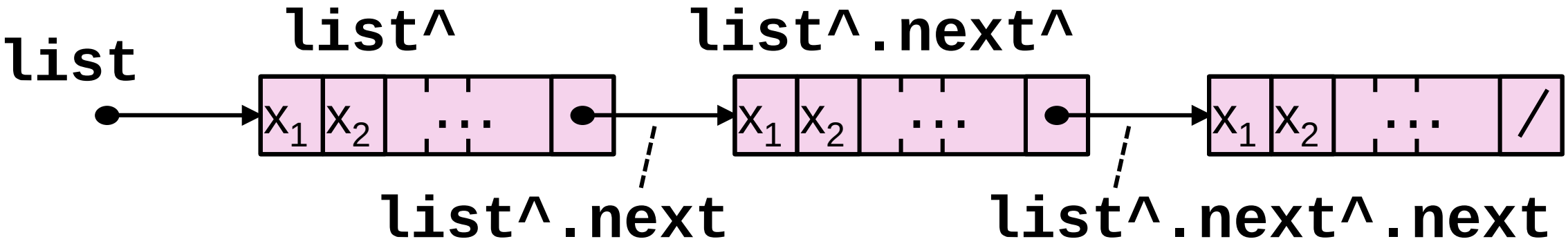
Linked Lists

- flexible data structure

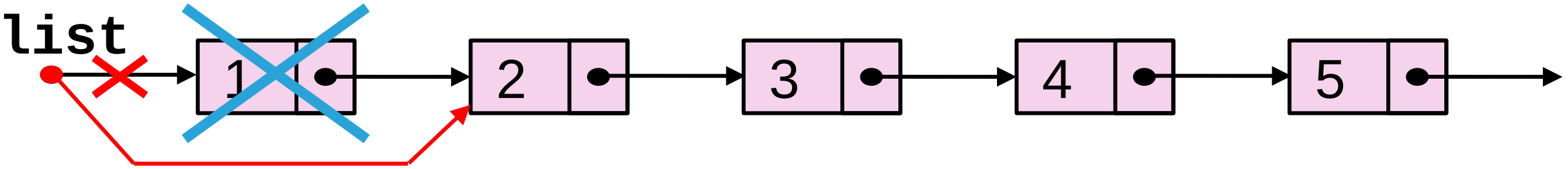


data

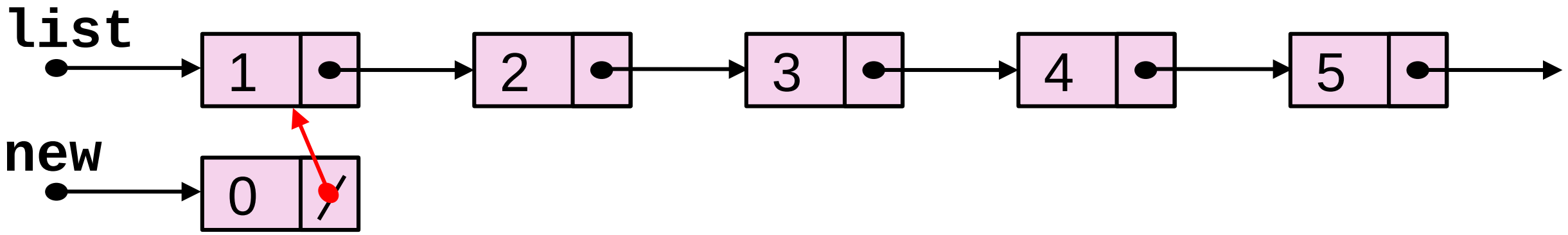
pointer
(link to next element)



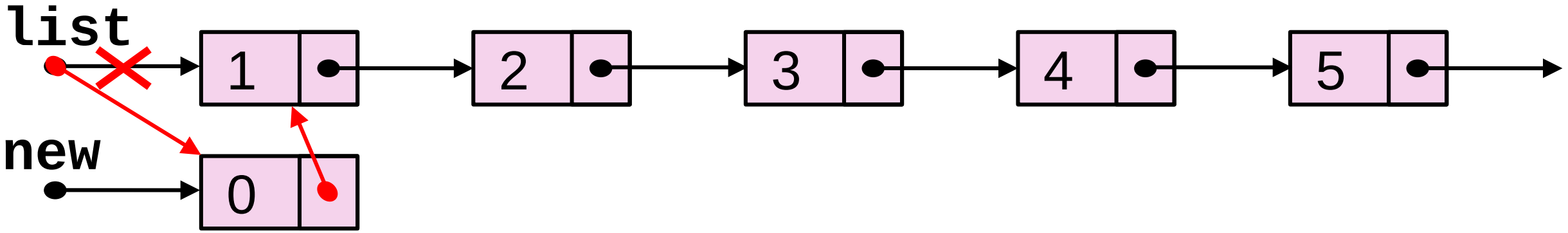
Linked Lists: Removal of First Element



Linked Lists: Inserting New First Element

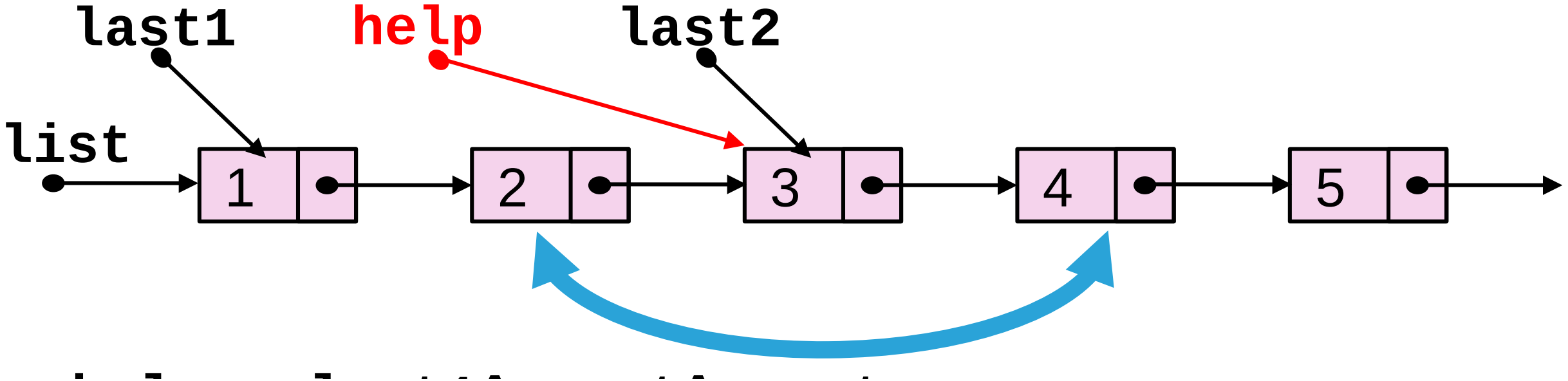


Linked Lists: Inserting New First Element

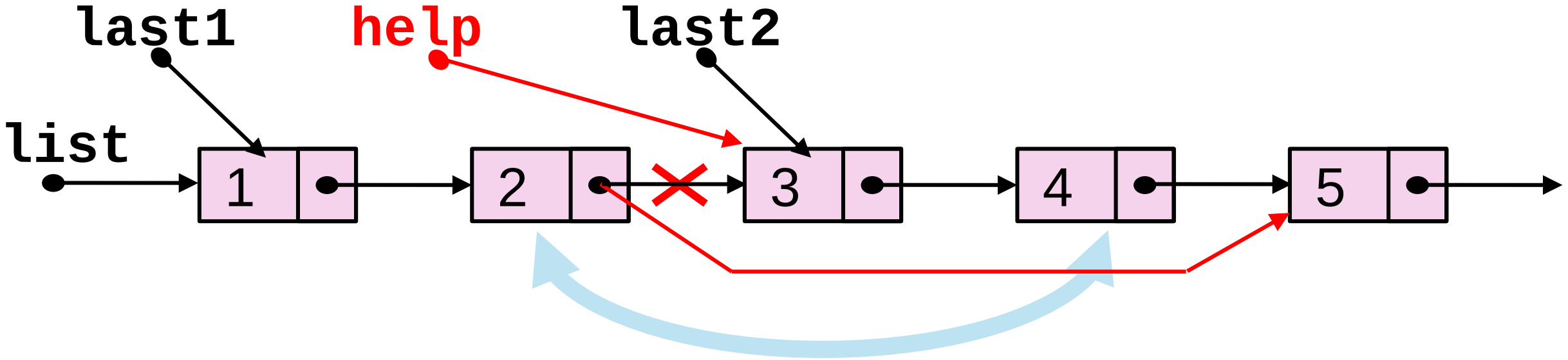



```
new^.next = list
```

Linked Lists: Exchanging Elements 2 & 4

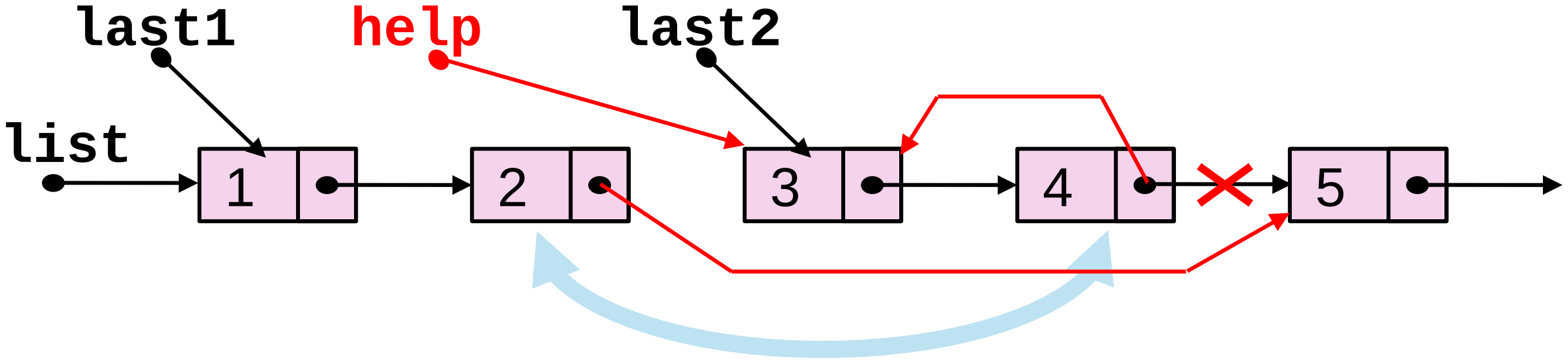


Linked Lists: Exchanging Elements 2 & 4



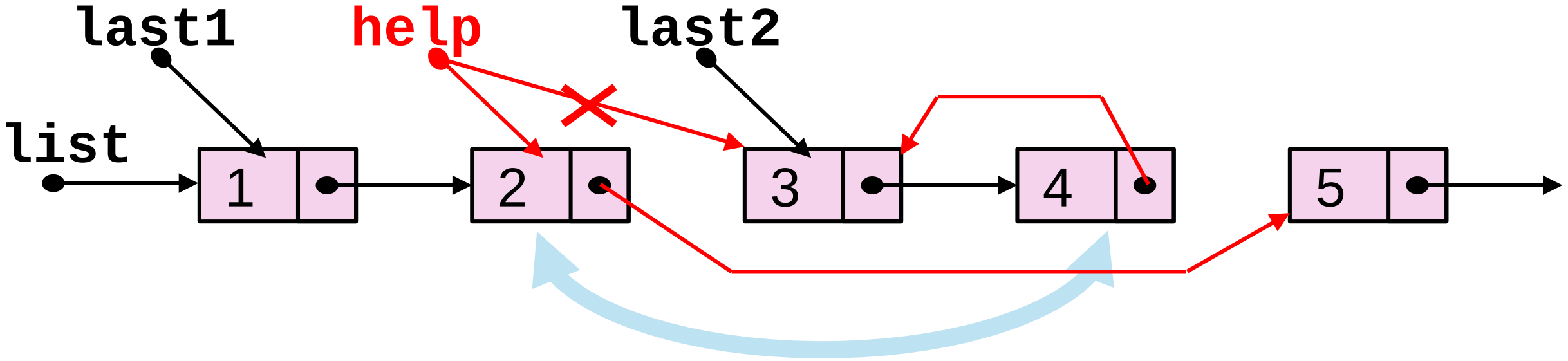

```
help = last1^.next^.next
```

Linked Lists: Exchanging Elements 2 & 4



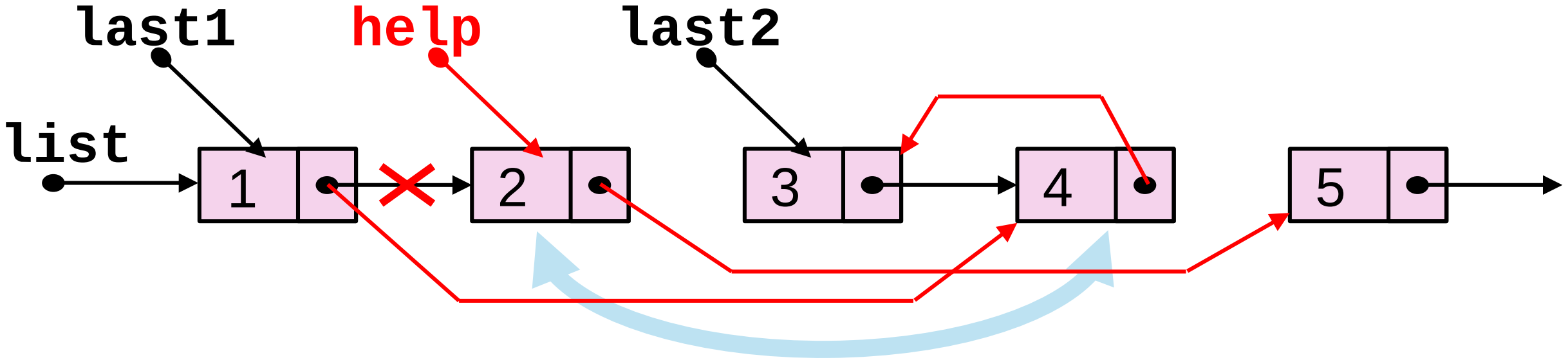

```
help = last1^.next^.next
last1^.next^.next = last2^.next^.next
```

Linked Lists: Exchanging Elements 2 & 4



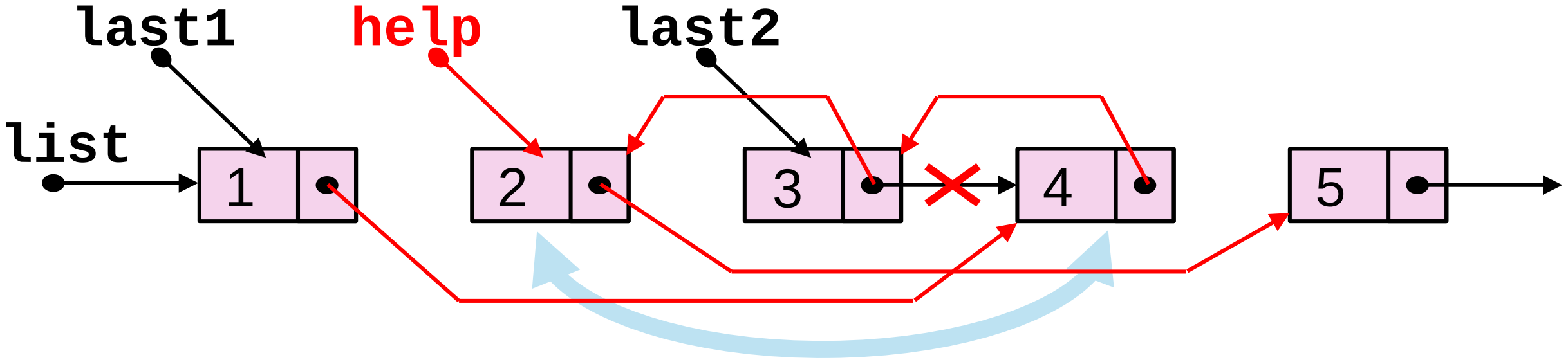

```
help = last1^.next^.next
last1^.next^.next = last2^.next^.next
last2^.next^.next = help
```

Linked Lists: Exchanging Elements 2 & 4



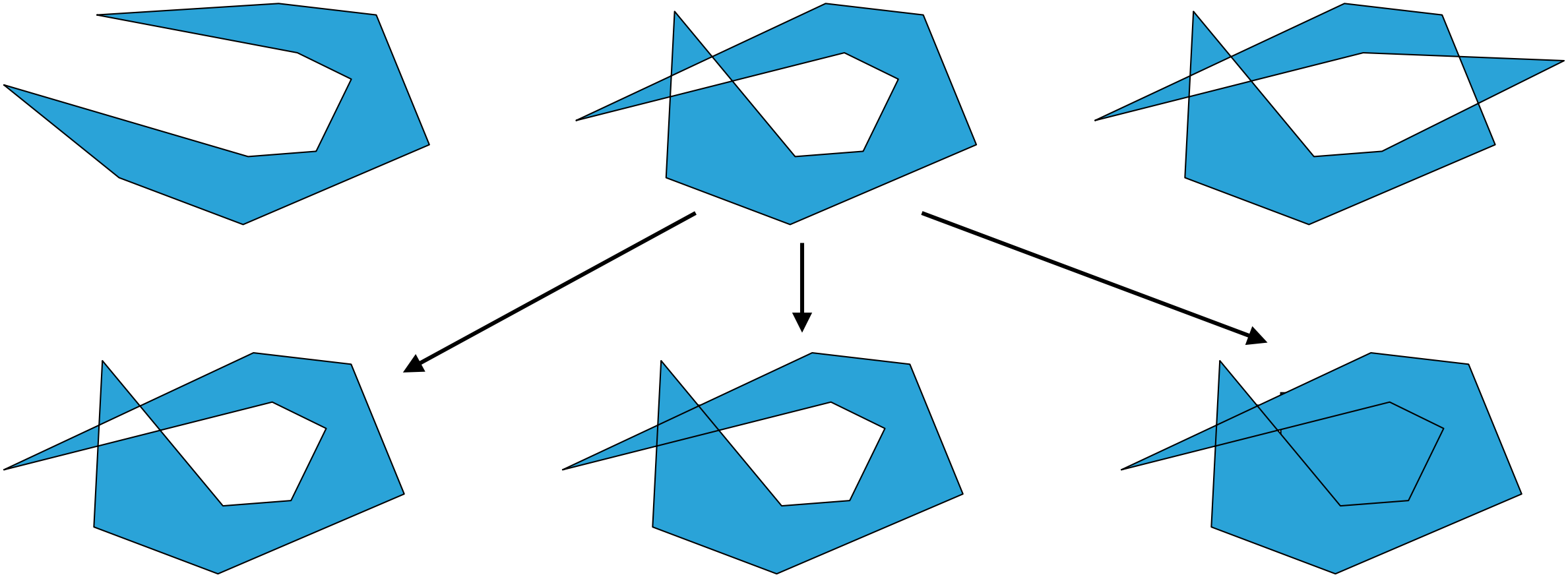

```
help = last1^.next^.next
last1^.next^.next = last2^.next^.next
last2^.next^.next = help
help = last1^.next
```

Linked Lists: Exchanging Elements 2 & 4




```
help = last1^.next^.next
last1^.next^.next = last2^.next^.next
last2^.next^.next = help
help = last1^.next
last1^.next = last2^.next
```

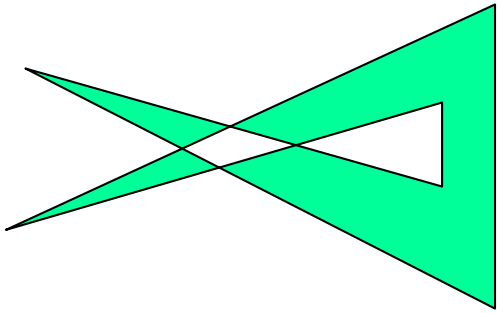
Repetition: What is Inside a Polygon?

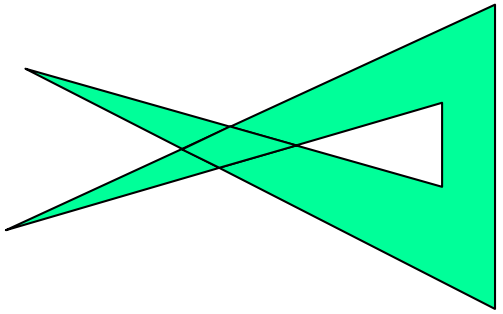


odd-even rule nonzero-winding- number rule ???

Inside-Outside Tests

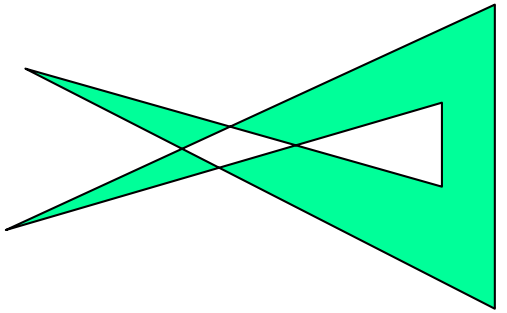
- area-filling algorithms
  - “interior”, “exterior” for self-intersecting polygons?
  - odd-even rule
  - nonzero-winding-number rule
  - same result for simple polygons

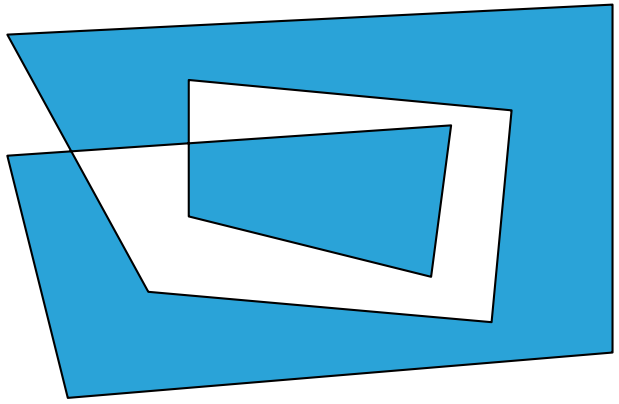


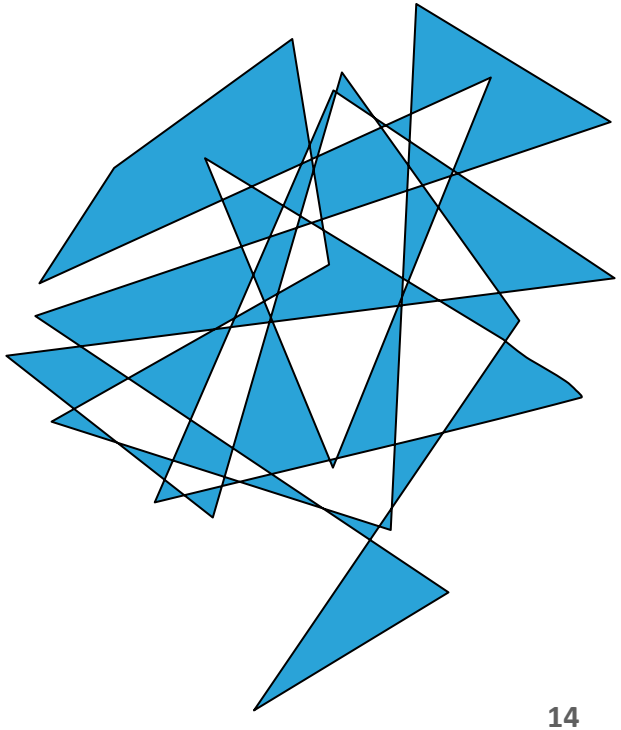


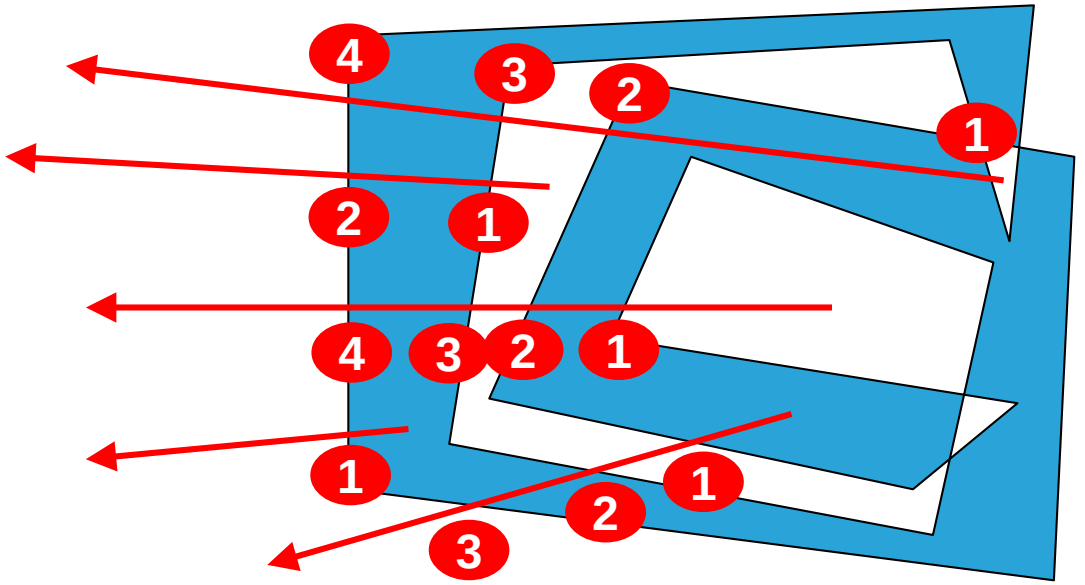
What is Inside?: Odd-Even Rule

- inside/outside switches at every edge
- straight line to the outside:
  - even # edge intersections = outside
  - odd # edge intersections = inside



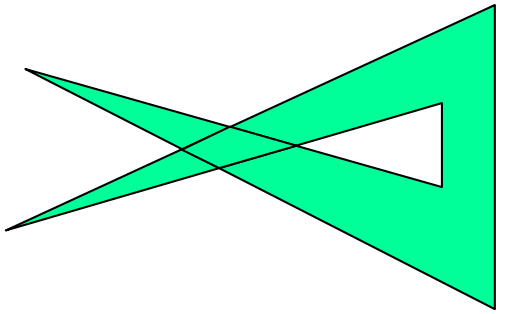


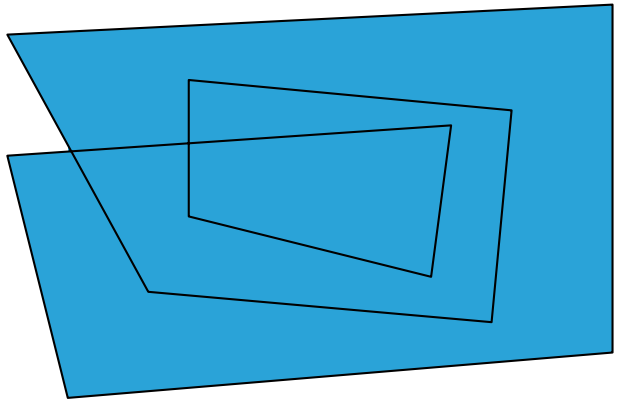


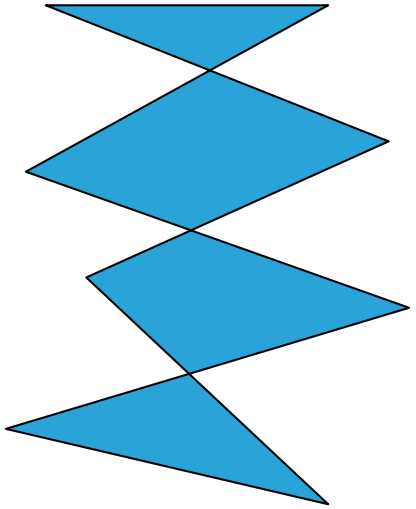


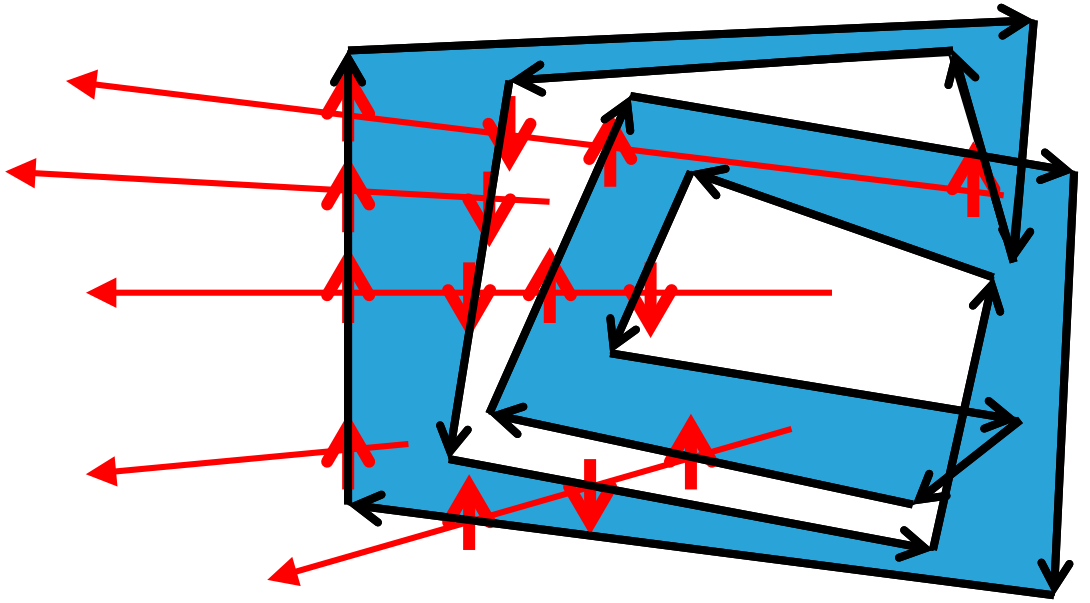
What is Inside?: Nonzero Winding Number

- point is inside if polygon surrounds it
- straight line to the outside:
  - same # edges up and down = outside
  - different # edges up and down = inside





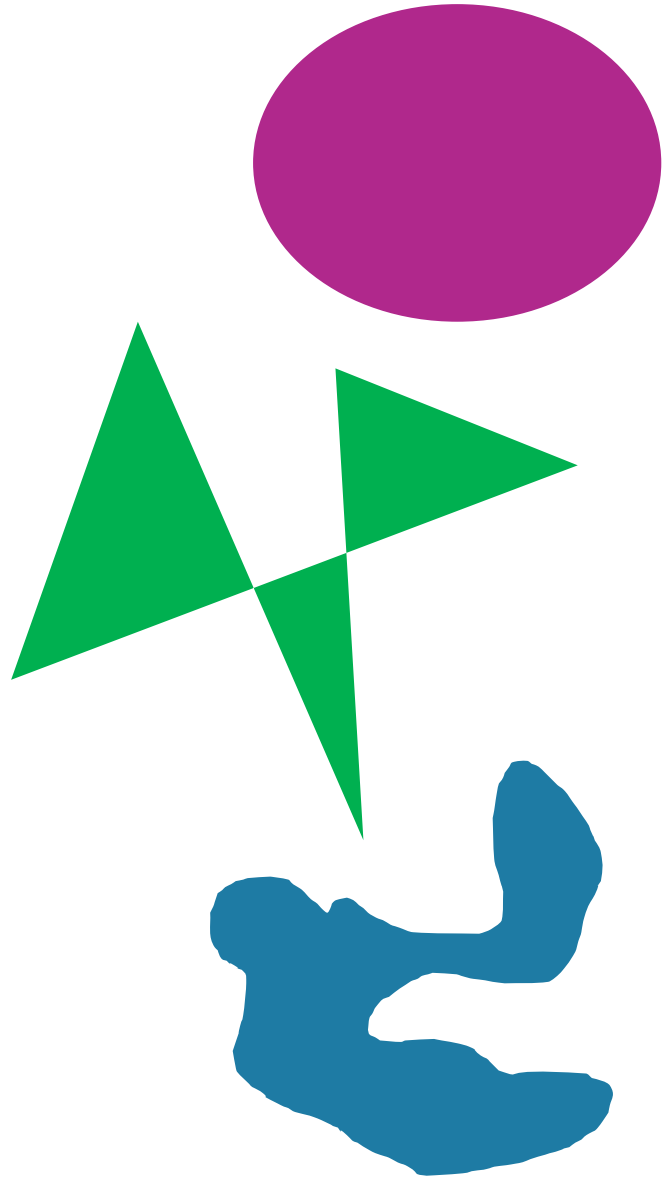




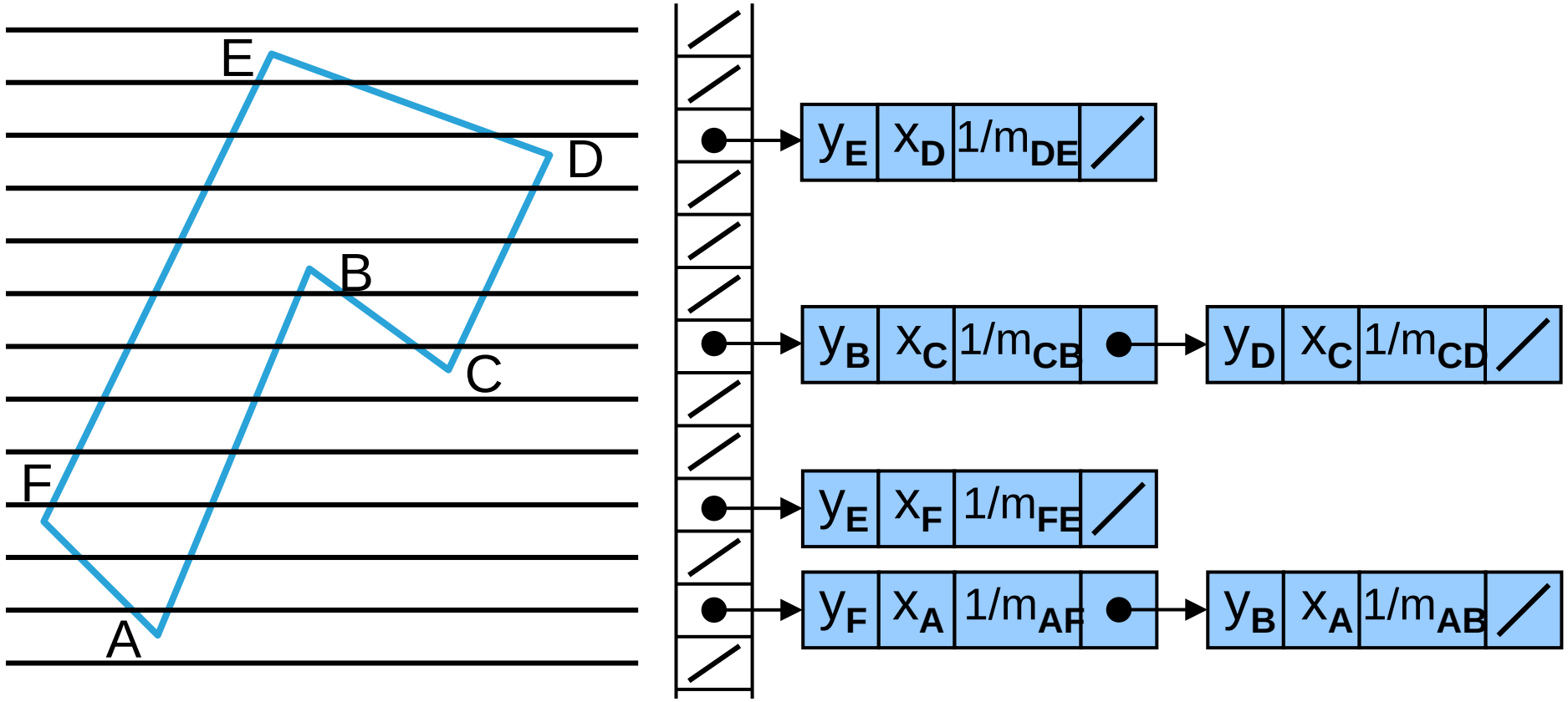
Fill-Area Primitives

for polygon area (solid-color, patterned)

- ***scan-line*** polygon fill algorithm
  - intersection points located and sorted
  - consecutive pairs define interior span
  - attention with vertex intersections
  - exploit coherence (incremental calculations)
- ***flood fill*** algorithm
- ***parallel*** algorithm
  - convex polygons only
  - check every pixel in bounding box



Scan-Line Fill: Sorted Edge Table

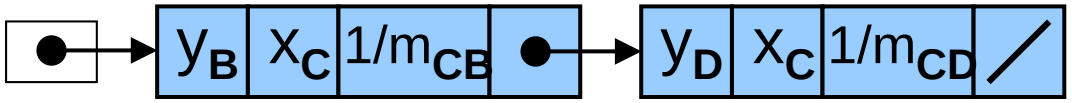


The sorted edge table contains all polygon edges sorted by lowest y-value

y_{max}	x_{start}	1/m	

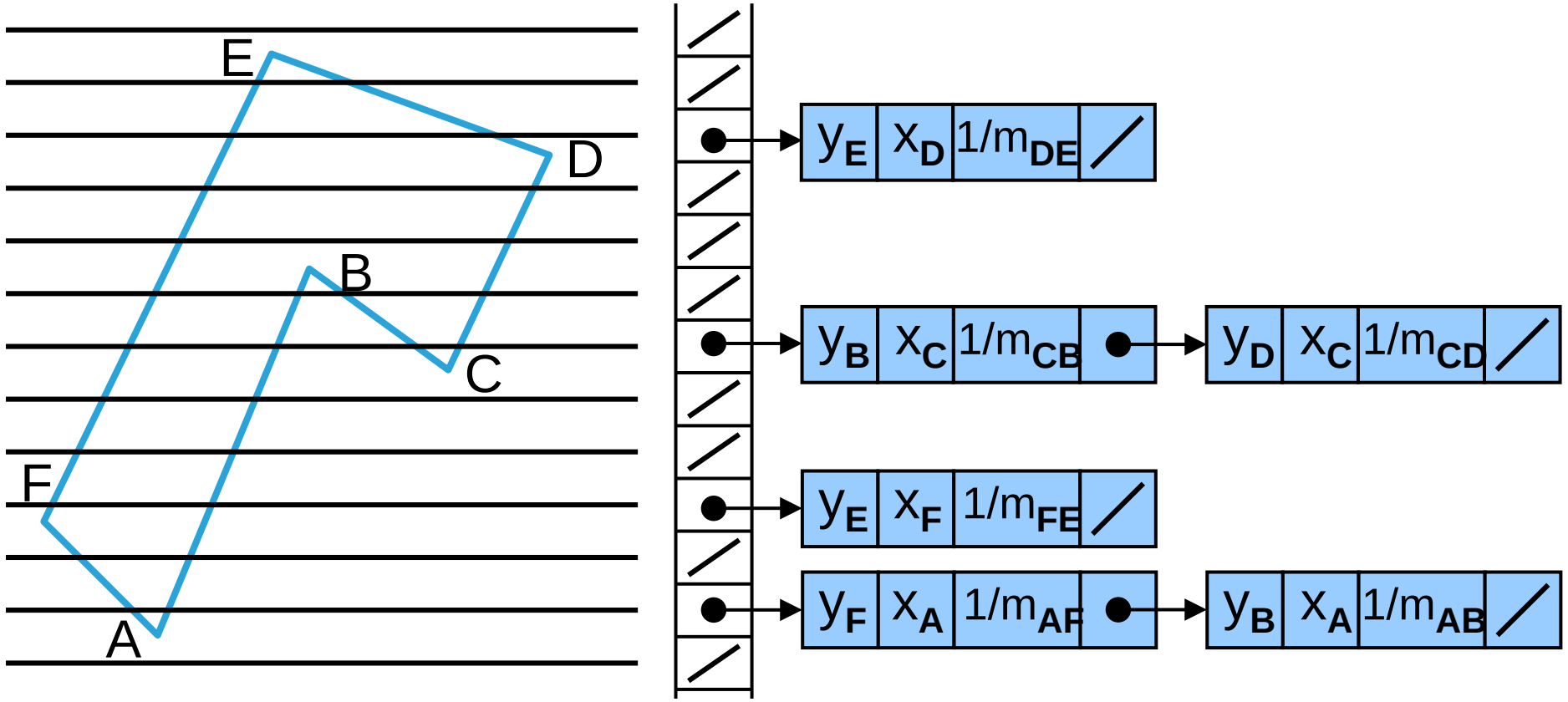
Scan-Line Fill: Sorted Edge Table

- sort all edges by smallest y-value
- edge entry:
  [max. y-value, x-intercept, inverse slope]



- ***active-edge list***
  - for each scan line
  - contains all edges crossed by that scan line
  - incremental update
- consecutive intersection pairs (spans) filled

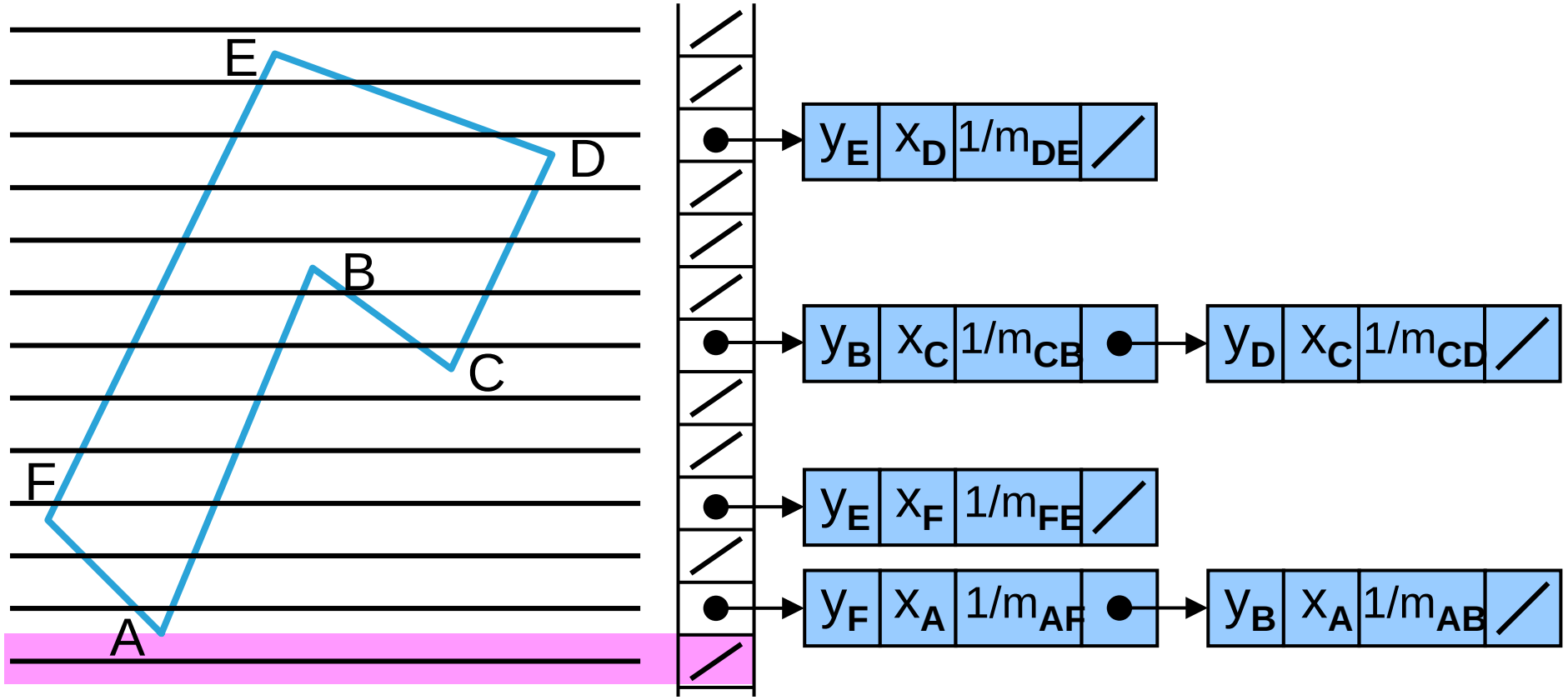
Sorted Edge Table



The sorted edge table contains all polygon edges sorted by lowest y-value

y_{max}	x_{start}	1/m	

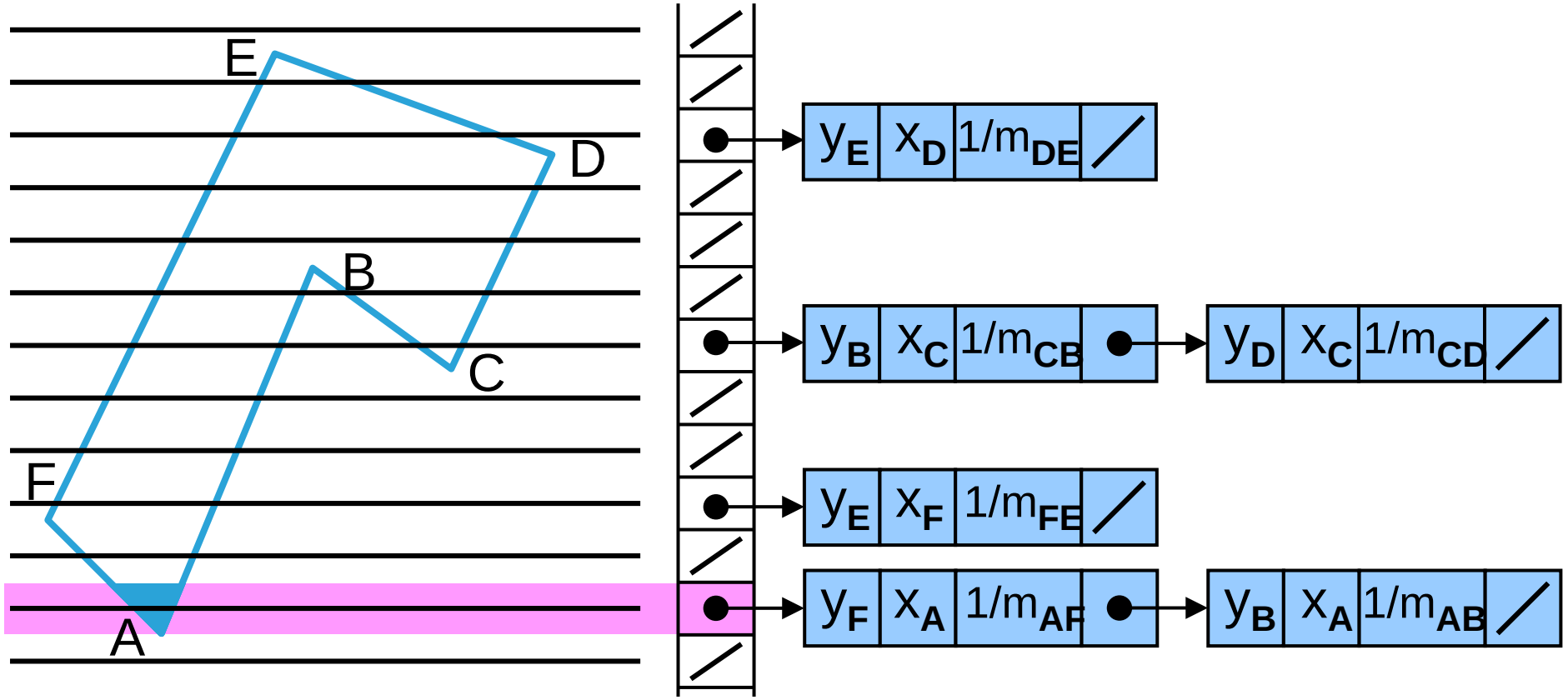
Sorted Edge Table / Active Edge List

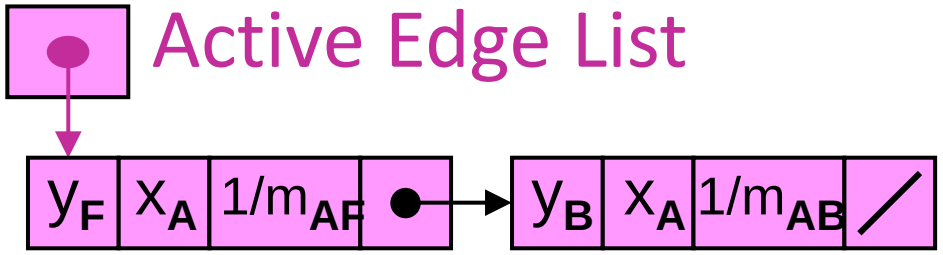


Active Edge List

When processing from bottom to top, keep a list of all active edges

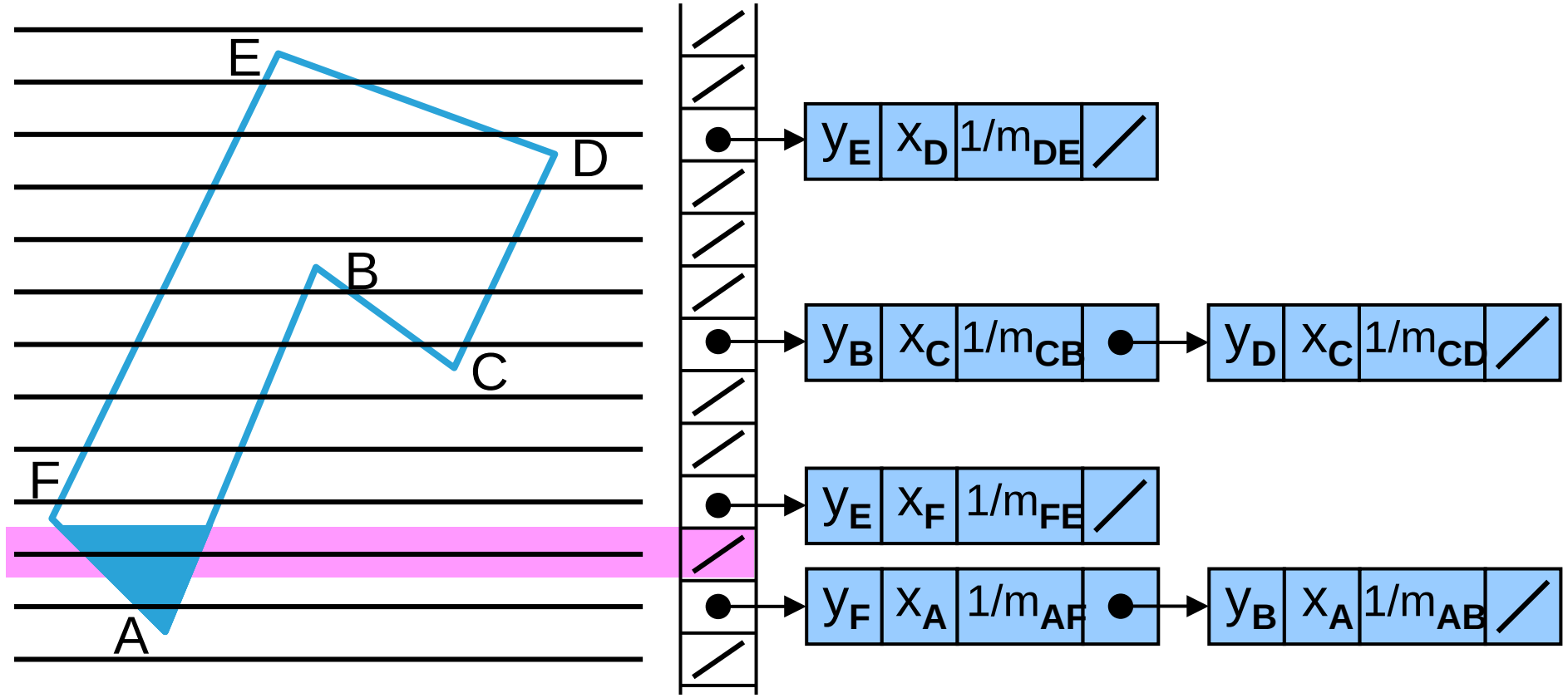
Sorted Edge Table / Active Edge List



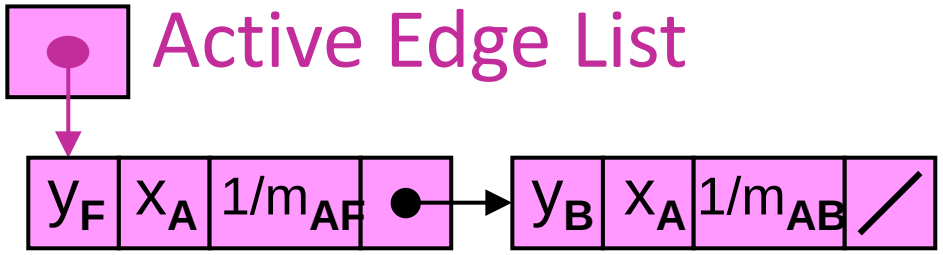


incremental update!

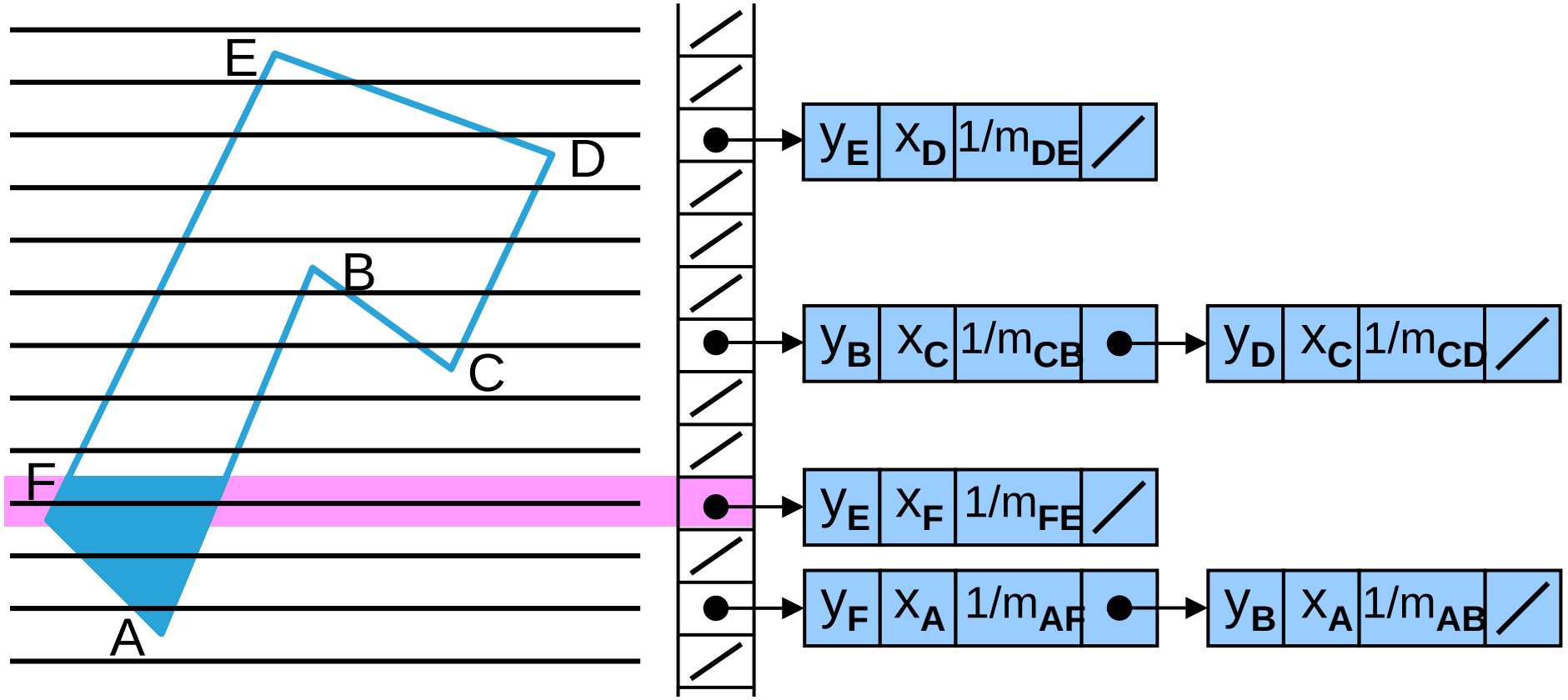
Sorted Edge Table / Active Edge List



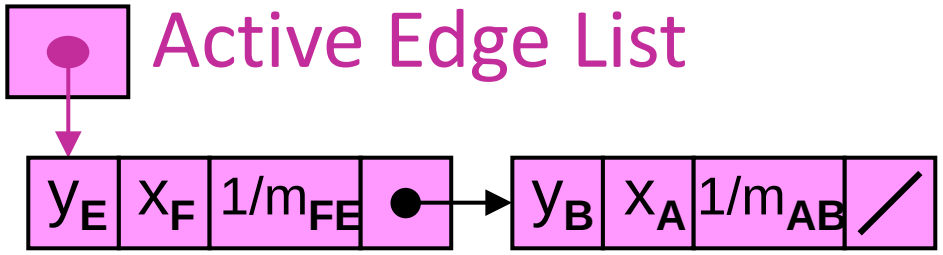
Active Edge List



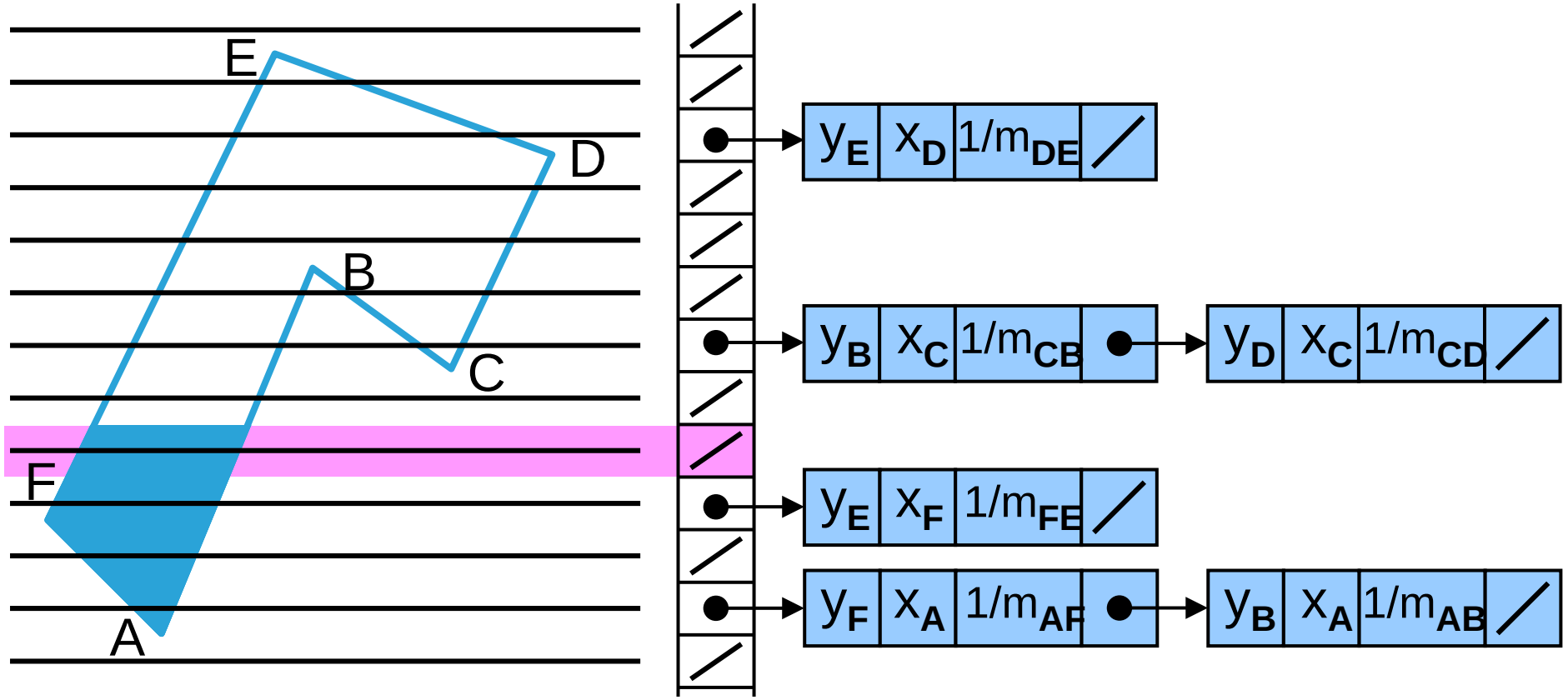
Sorted Edge Table / Active Edge List



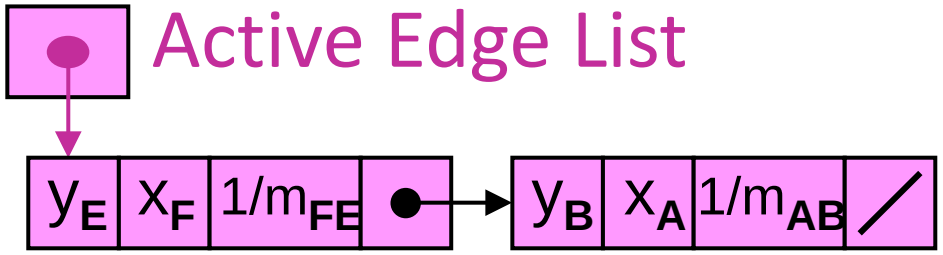
Active Edge List



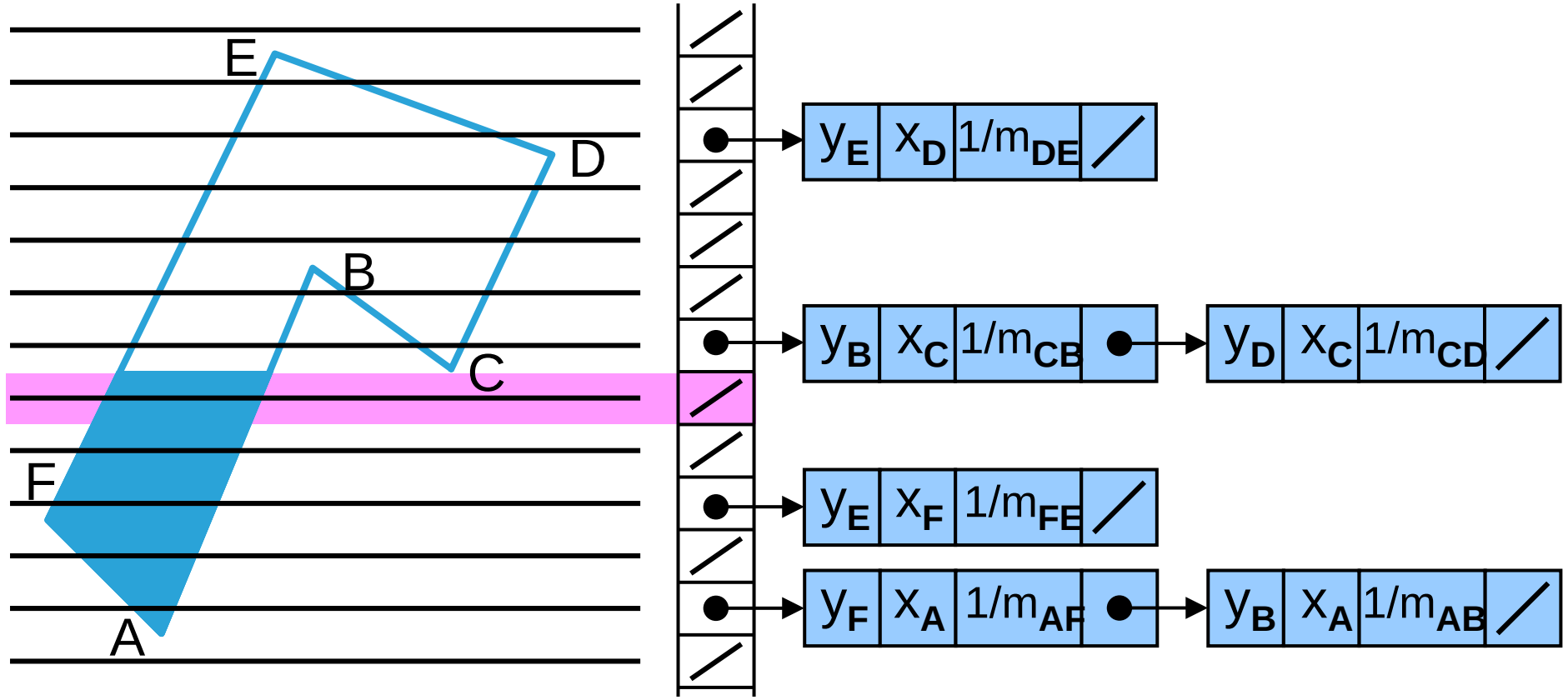
Sorted Edge Table / Active Edge List



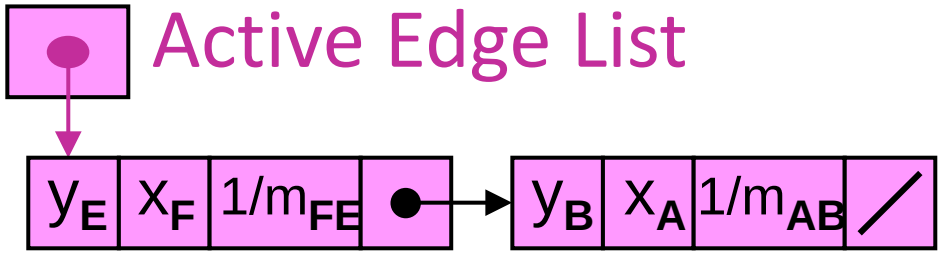
Active Edge List



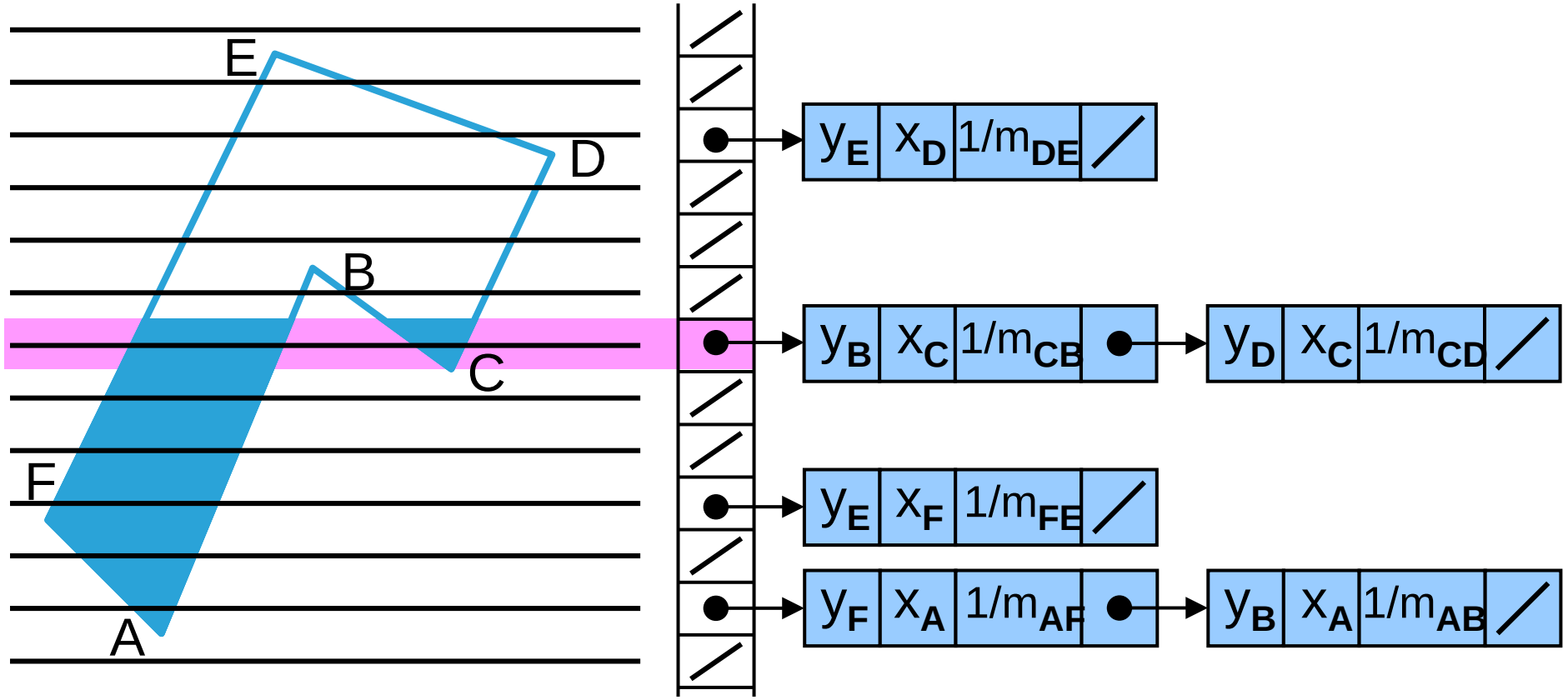
Sorted Edge Table / Active Edge List



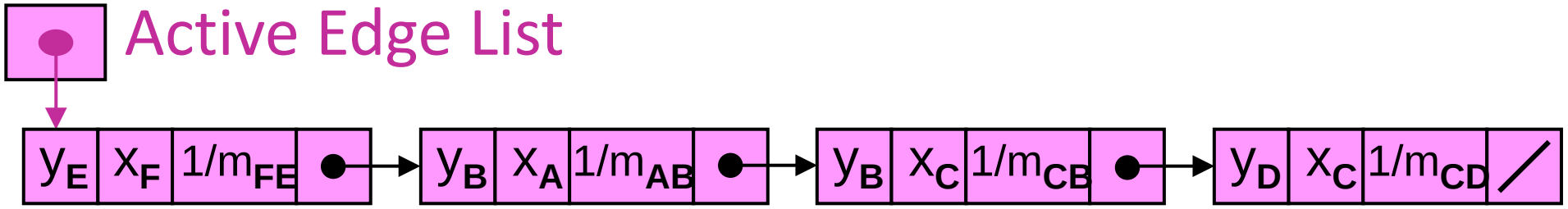
Active Edge List



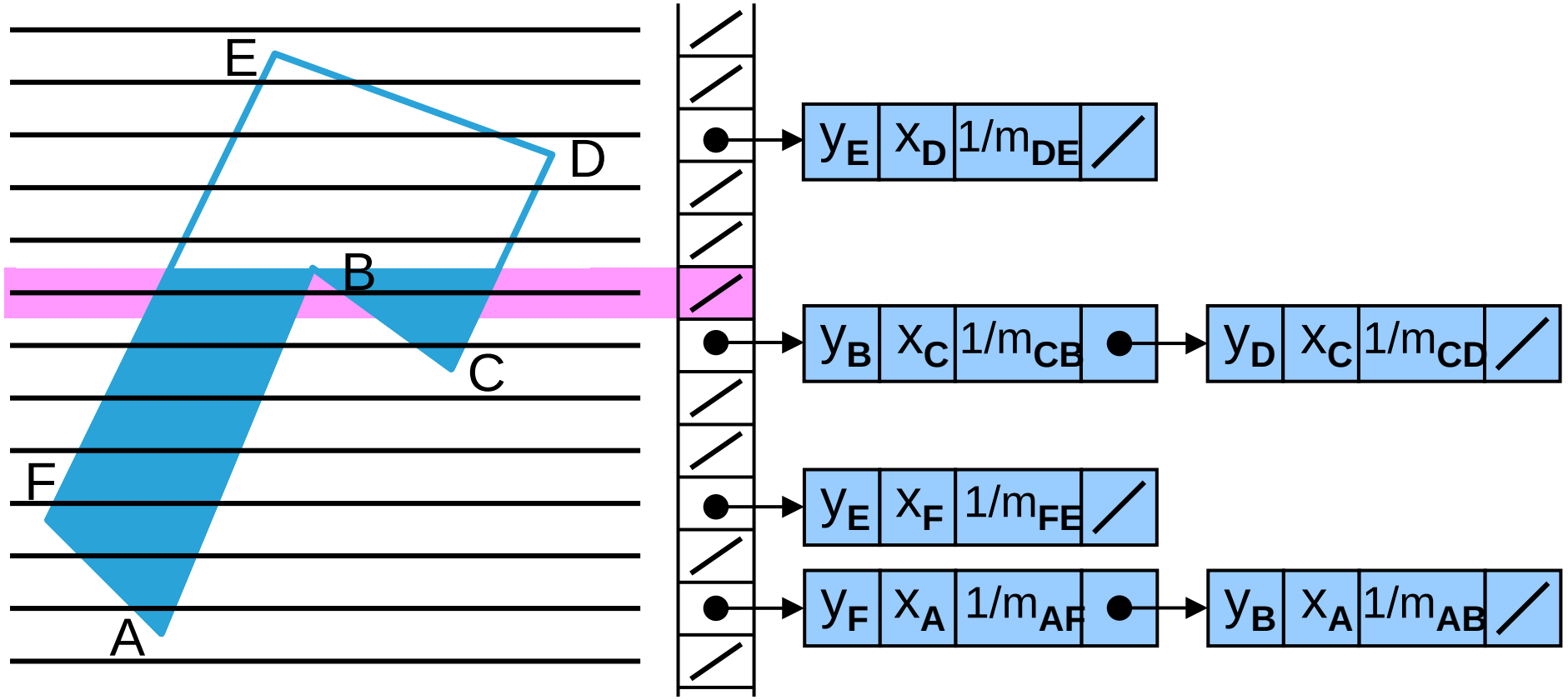
Sorted Edge Table / Active Edge List



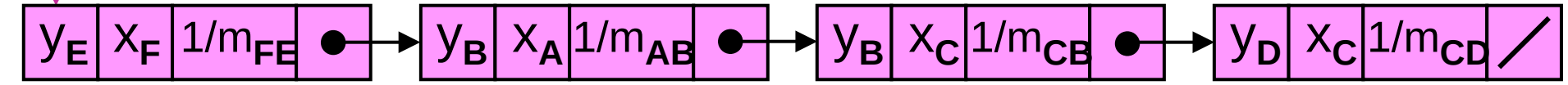
Active Edge List



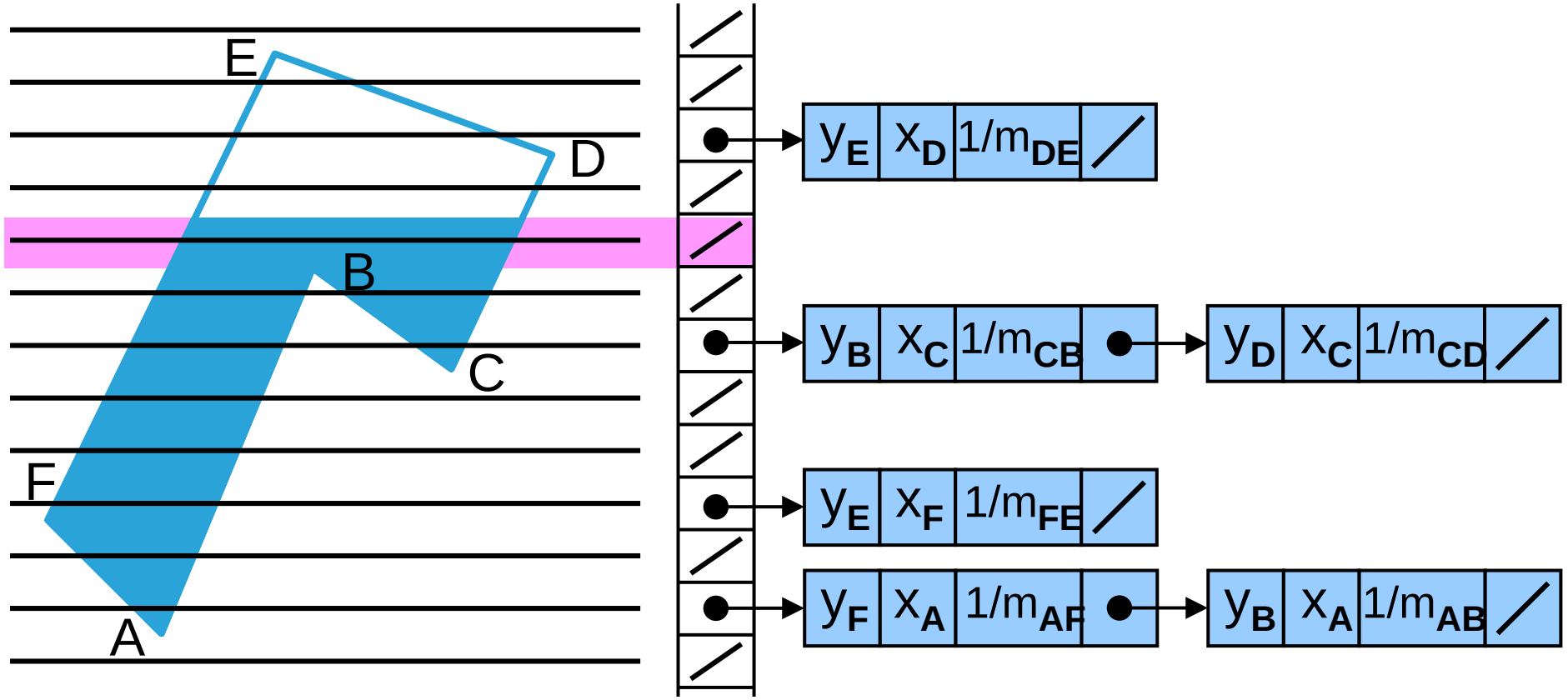
Sorted Edge Table / Active Edge List



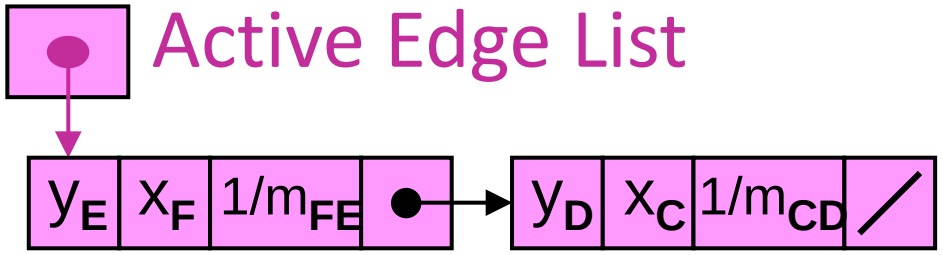
Active Edge List



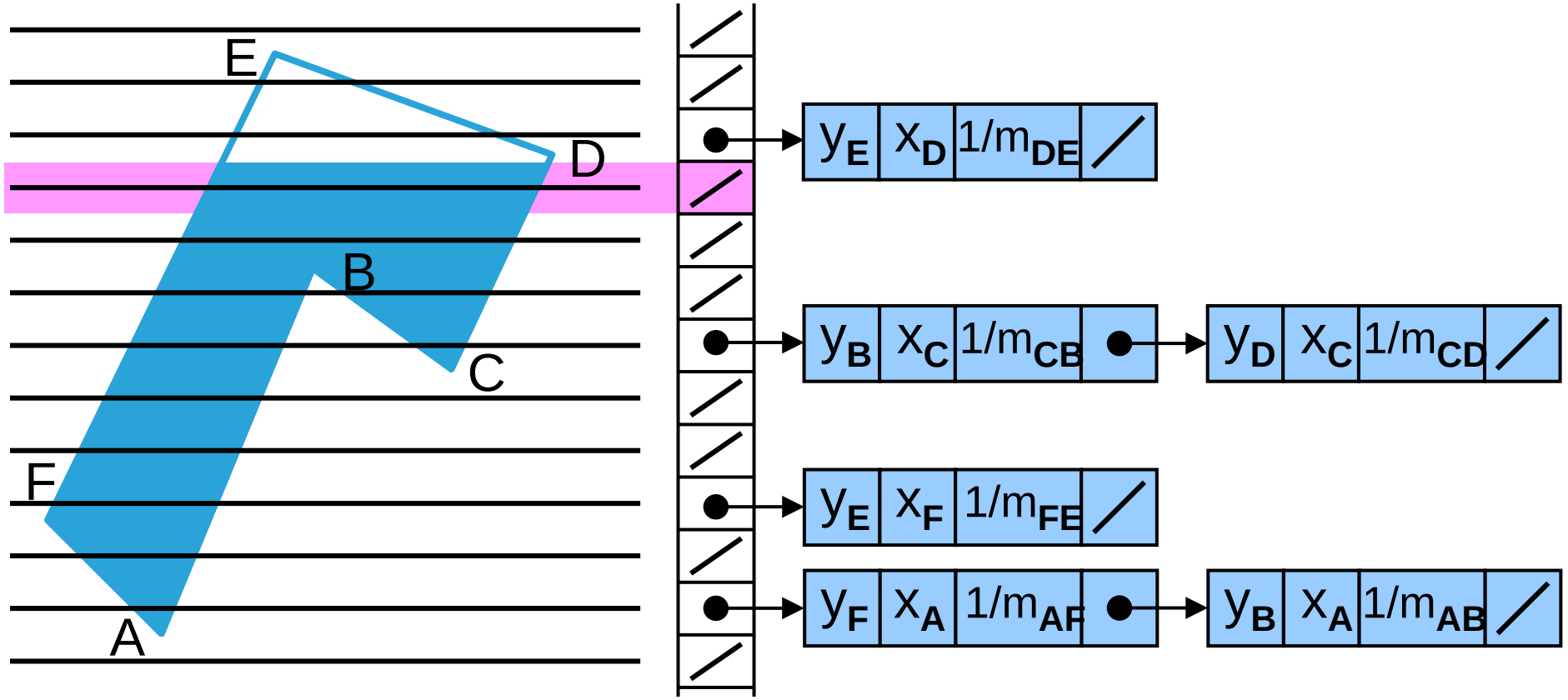
Sorted Edge Table / Active Edge List



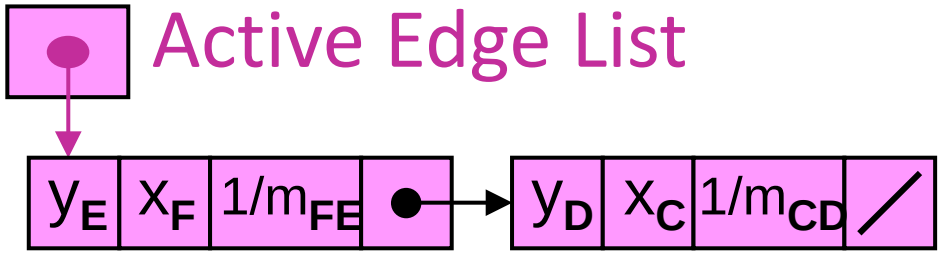
Active Edge List



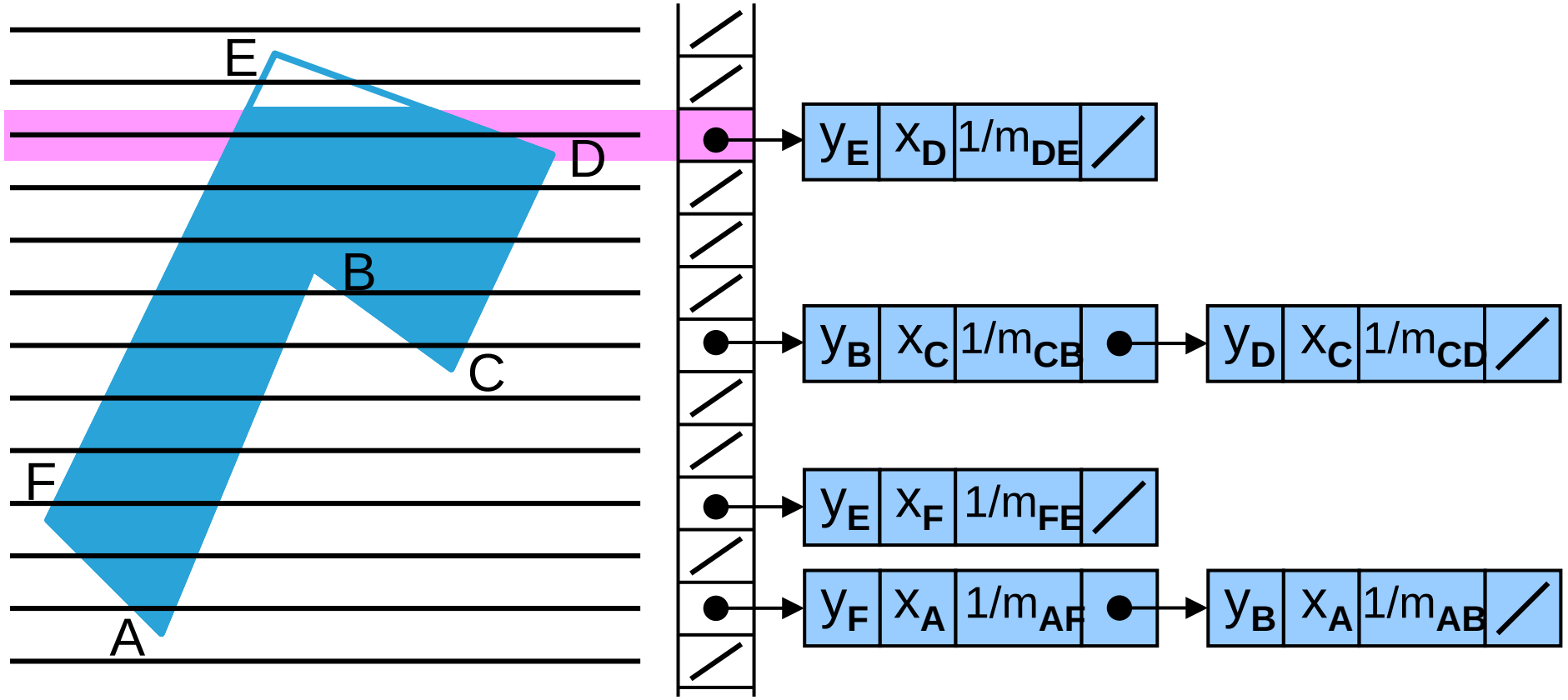
Sorted Edge Table / Active Edge List



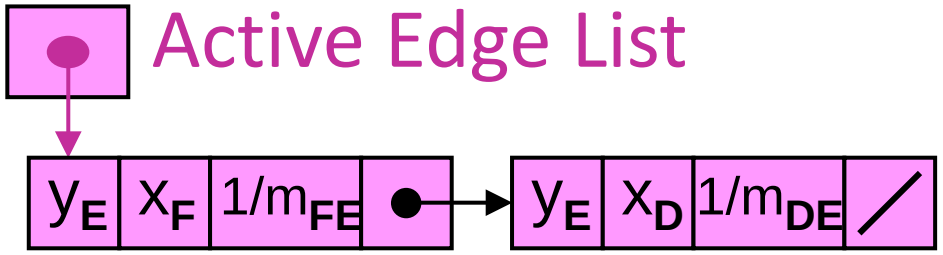
Active Edge List



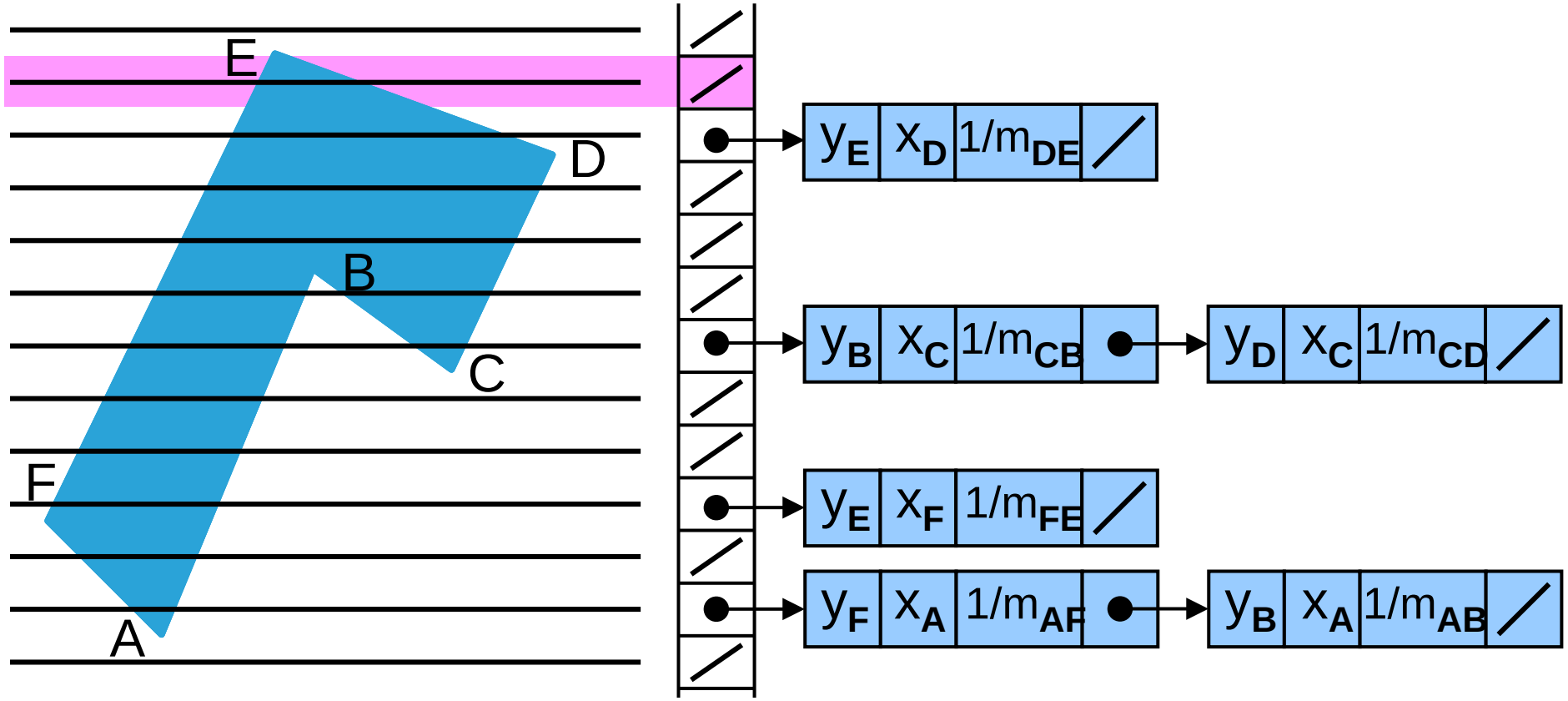
Sorted Edge Table / Active Edge List



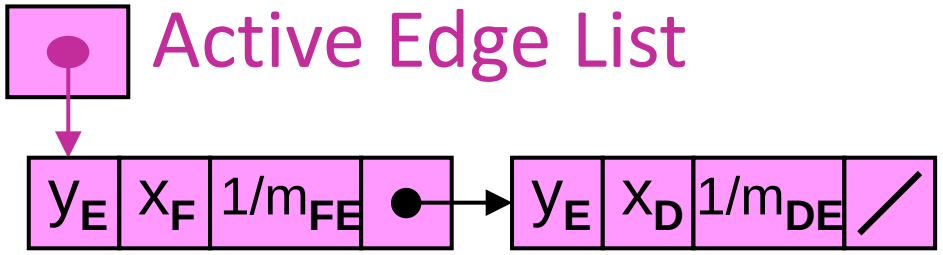
Active Edge List



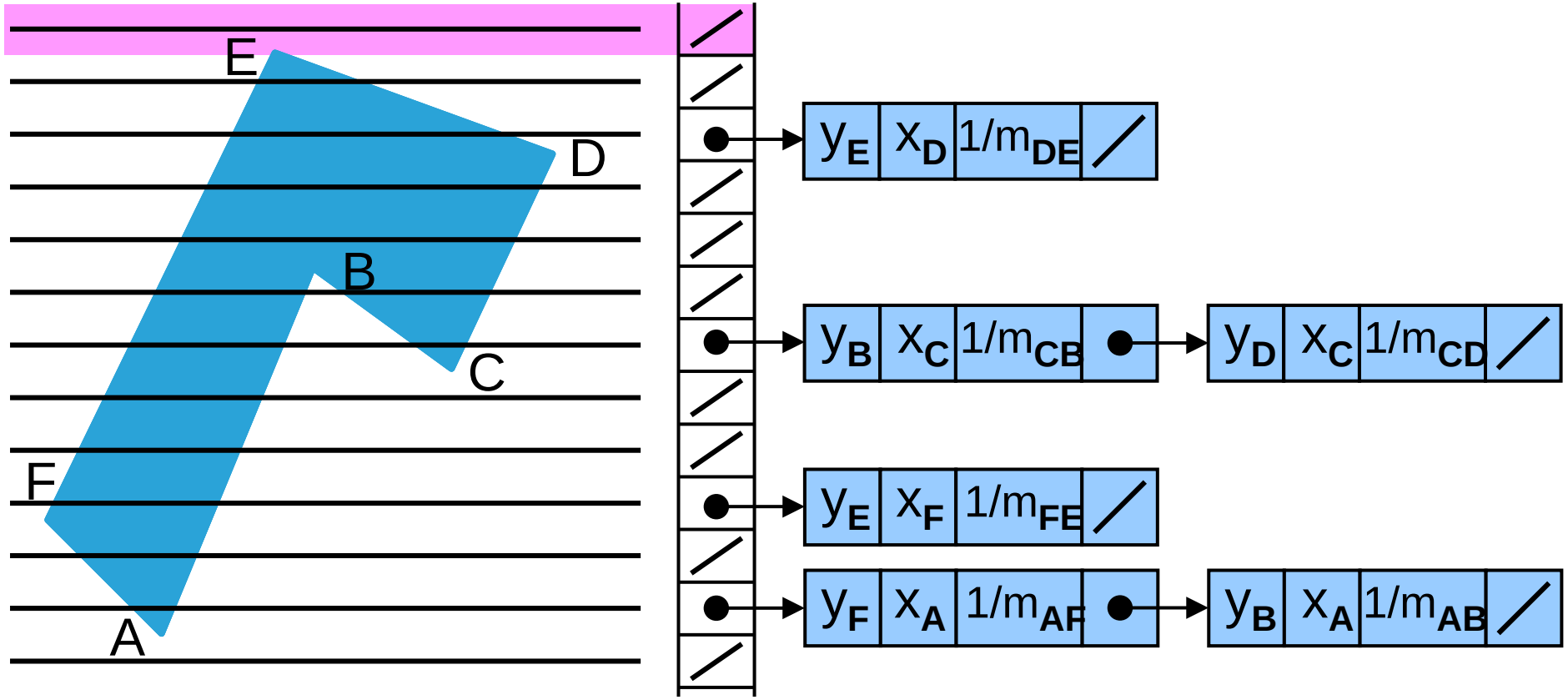
Sorted Edge Table / Active Edge List



Active Edge List



Sorted Edge Table / Active Edge List



Active Edge List

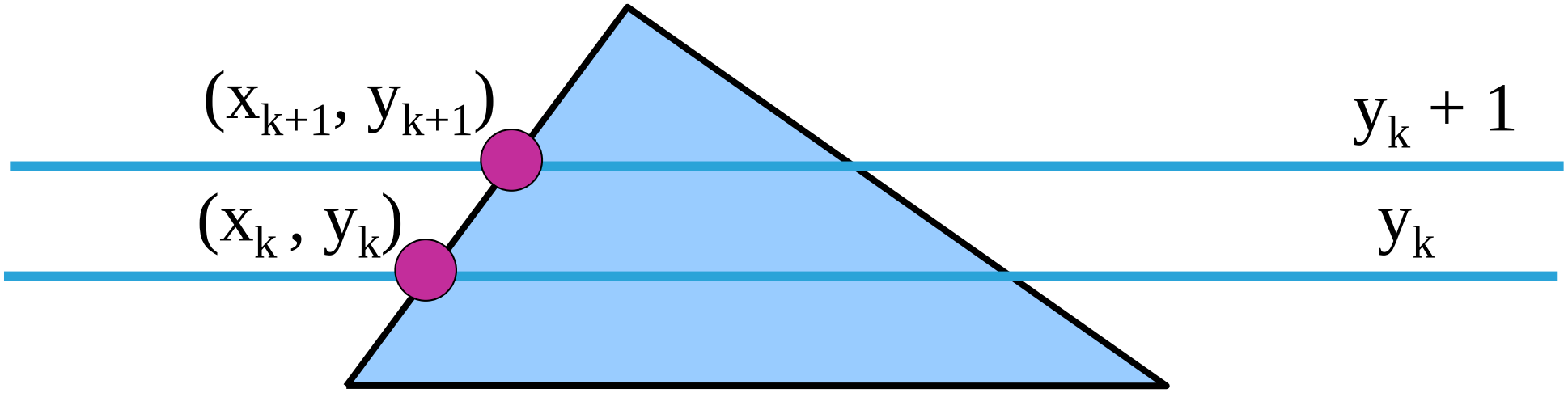
Scan-Line Fill: Incremental Update

- incremental update of intersection point

slope of polygon boundary line: m

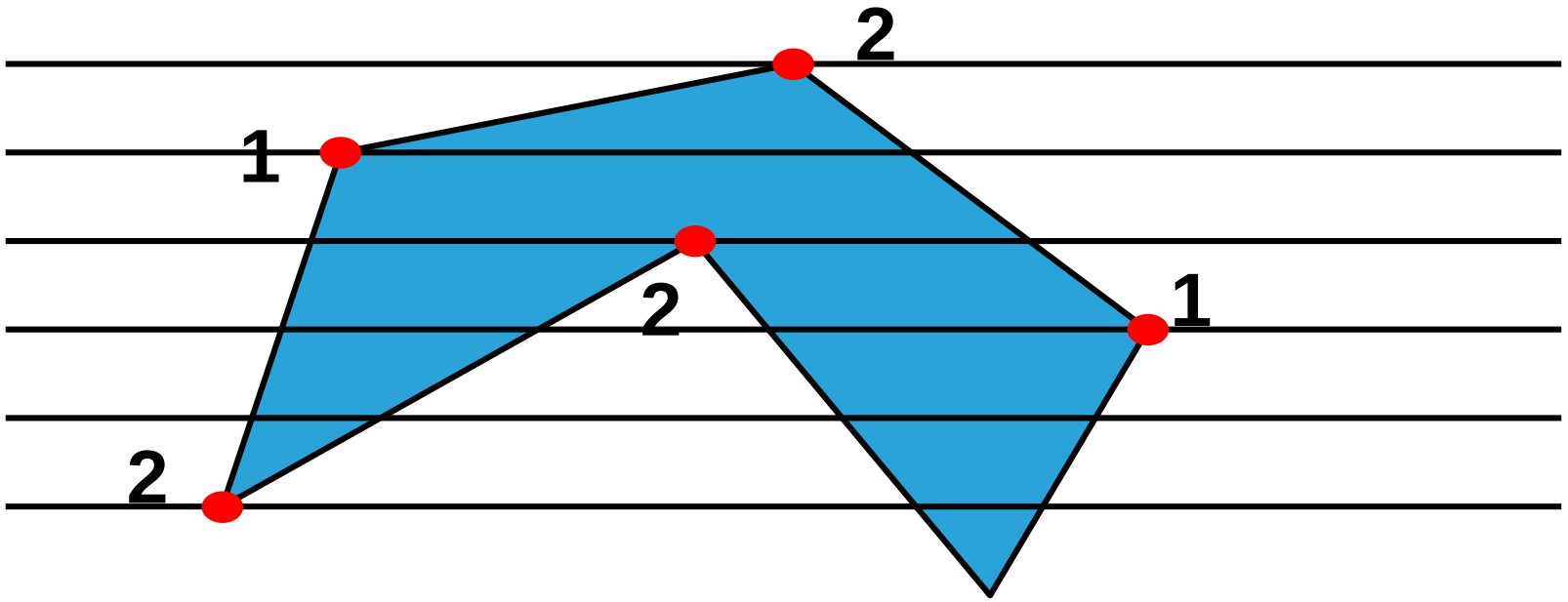
$$x_{k+1} = x_k + \frac{1}{m} \qquad y_{k+1} = y_k + 1$$

(for 2 successive scanlines)



Scan-Line Fill: Intersecting Vertices

intersection points along scan lines that intersect polygon vertices



→ either special handling (1 or 2 intersections?)

→ or move vertices up or down by ε

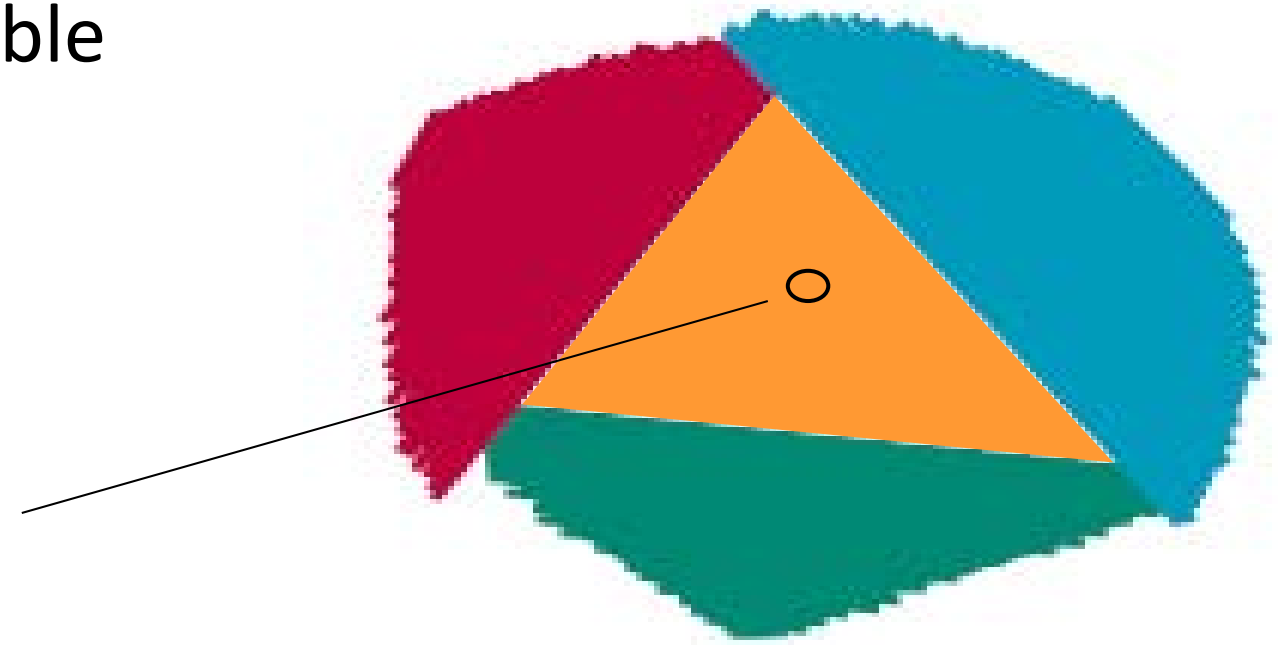
Flood-Fill Algorithm

- pixel filling of area
  - areas with no single color boundary
  - start from interior point
  - “flood” internal region
  - 4-connected, 8-connected areas
  - reduce stack size by eliminating several recursive calls

Flood-Fill: Boundary and Seed Point

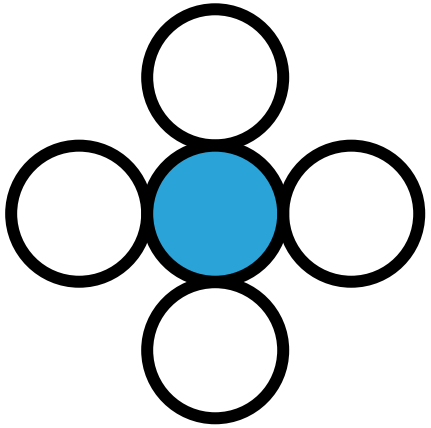
area must be distinguishable from boundaries

seed point

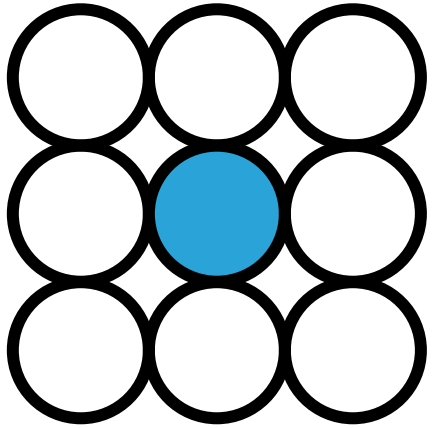


example: area defined within multiple color boundaries

Flood-Fill: Connectedness

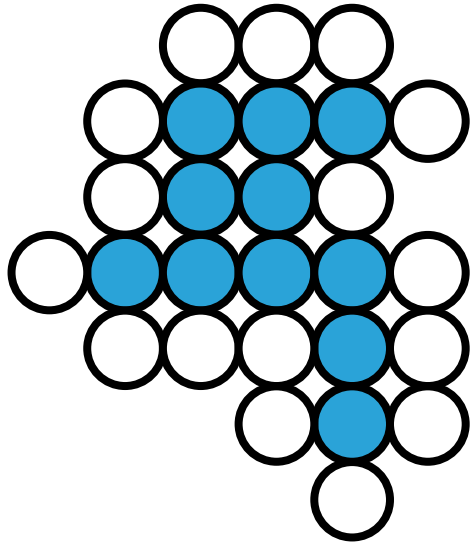


Definition:
4-connected means, that a connection is only valid in these 4 directions



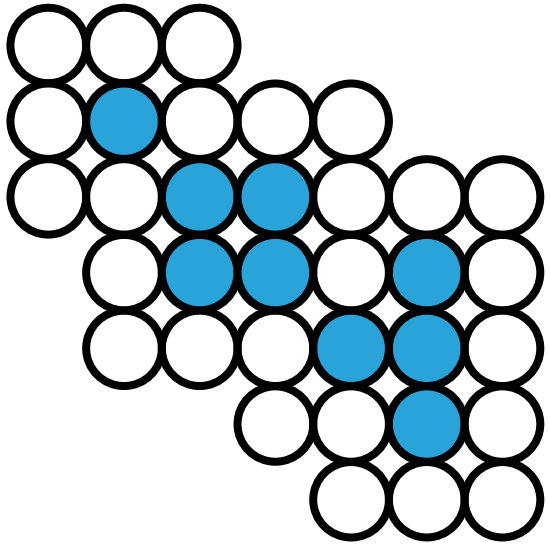
Definition:
8-connected means, that a connection is valid in these 8 directions

Examples for 4-connected and 8-connected



a 4-connected area

has an 8-connected border

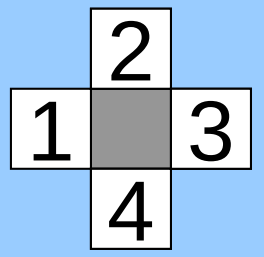


an 8-connected area

has a 4-connected border

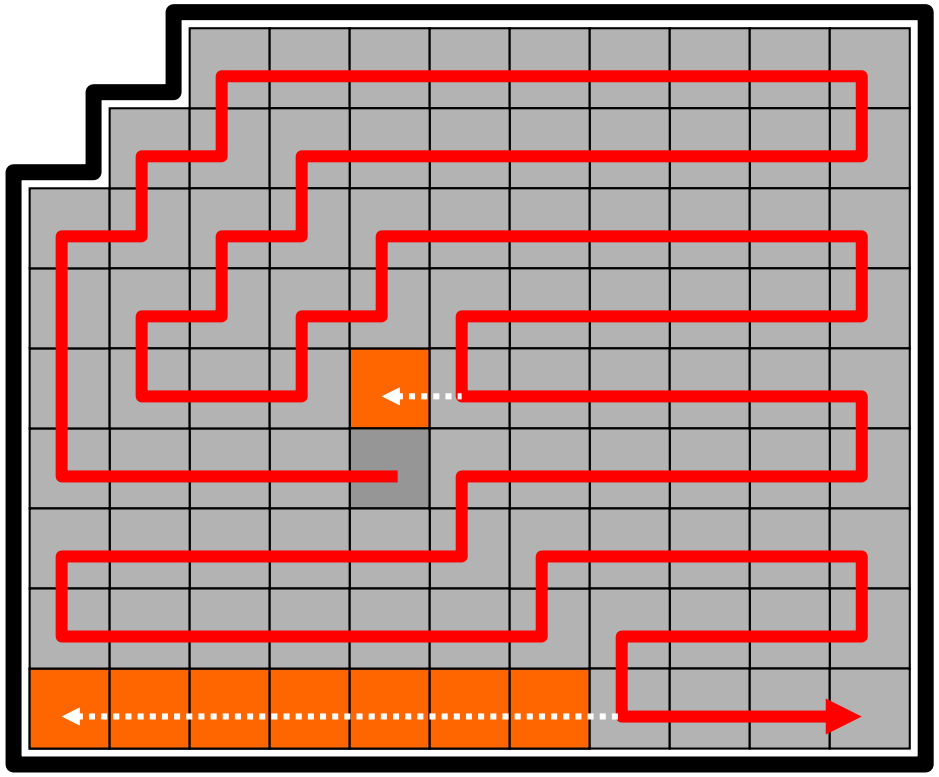
Simple Flood-Fill Algorithm

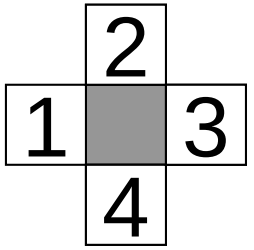
```
void floodFill4 (int x, int y, int new, int old)
{ int color;
    /* set current color to new */
    getPixel (x, y, color);
    if (color = old) {
        setPixel (x, y);
        floodFill4 (x–1, y, new, old);  /* left */
        floodFill4 (x, y+1, new, old);  /* up */
        floodFill4 (x+1, y, new, old);  /* right */
        floodFill4 (x, y–1, new, old)   /* down */
}}
```



recursion sequence

Bad Behavior of Simple Flood-Fill



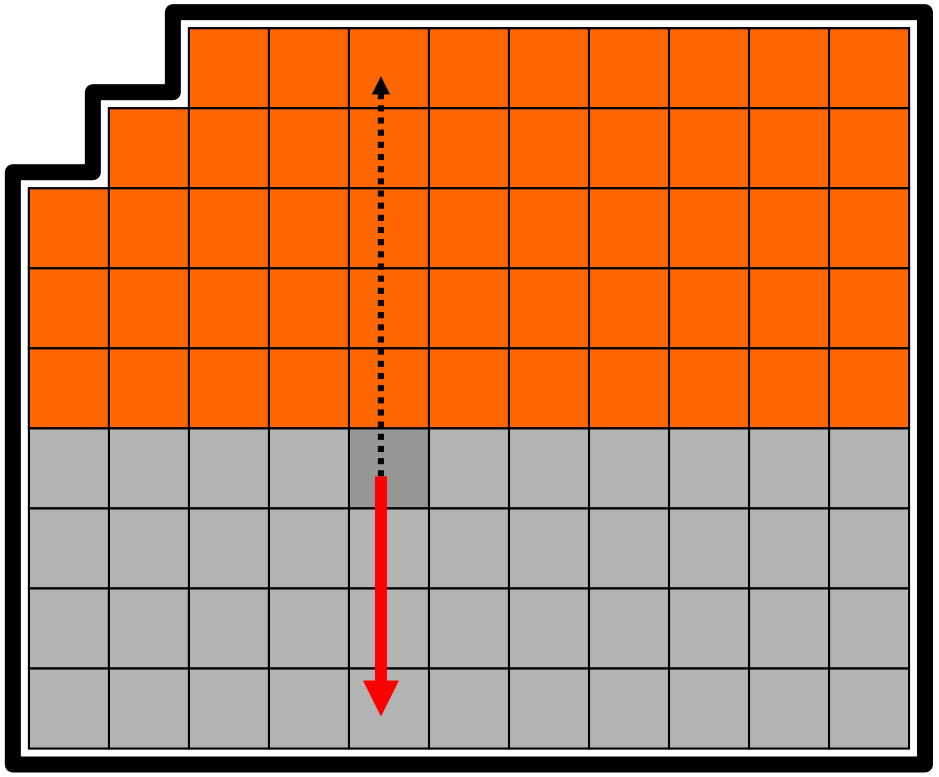


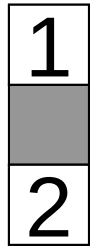
recursion sequence

Span Flood-Fill Algorithm

- floodFill4 produces too high stacks (recursion!)
- solution:
  - incremental horizontal fill (left to right)
  - recursive vertical fill (first up then down)

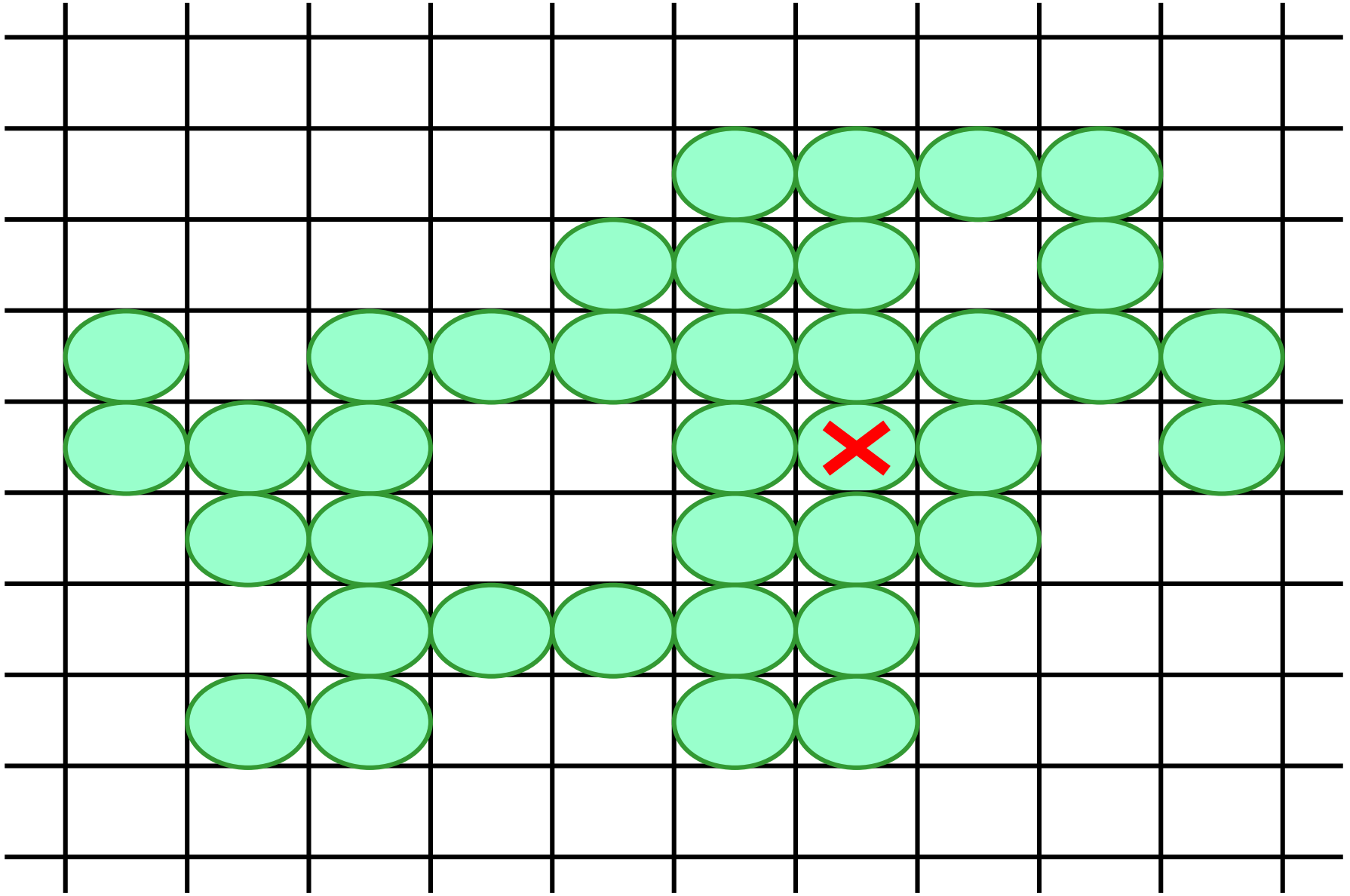
Good Behavior of Span Flood-Fill





recursion sequence

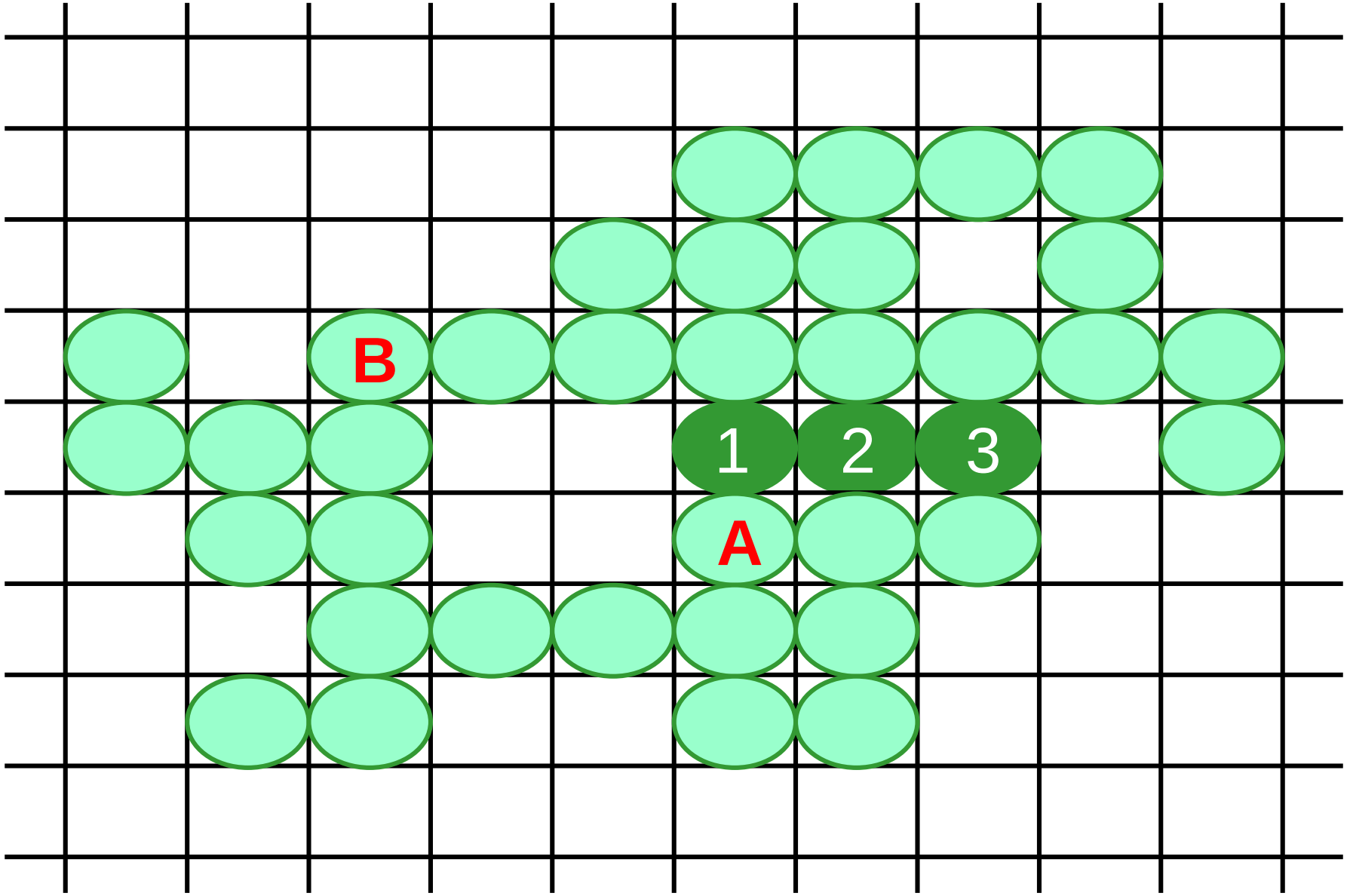
Span Flood-Fill Example



Stack:

x

Span Flood-Fill Example



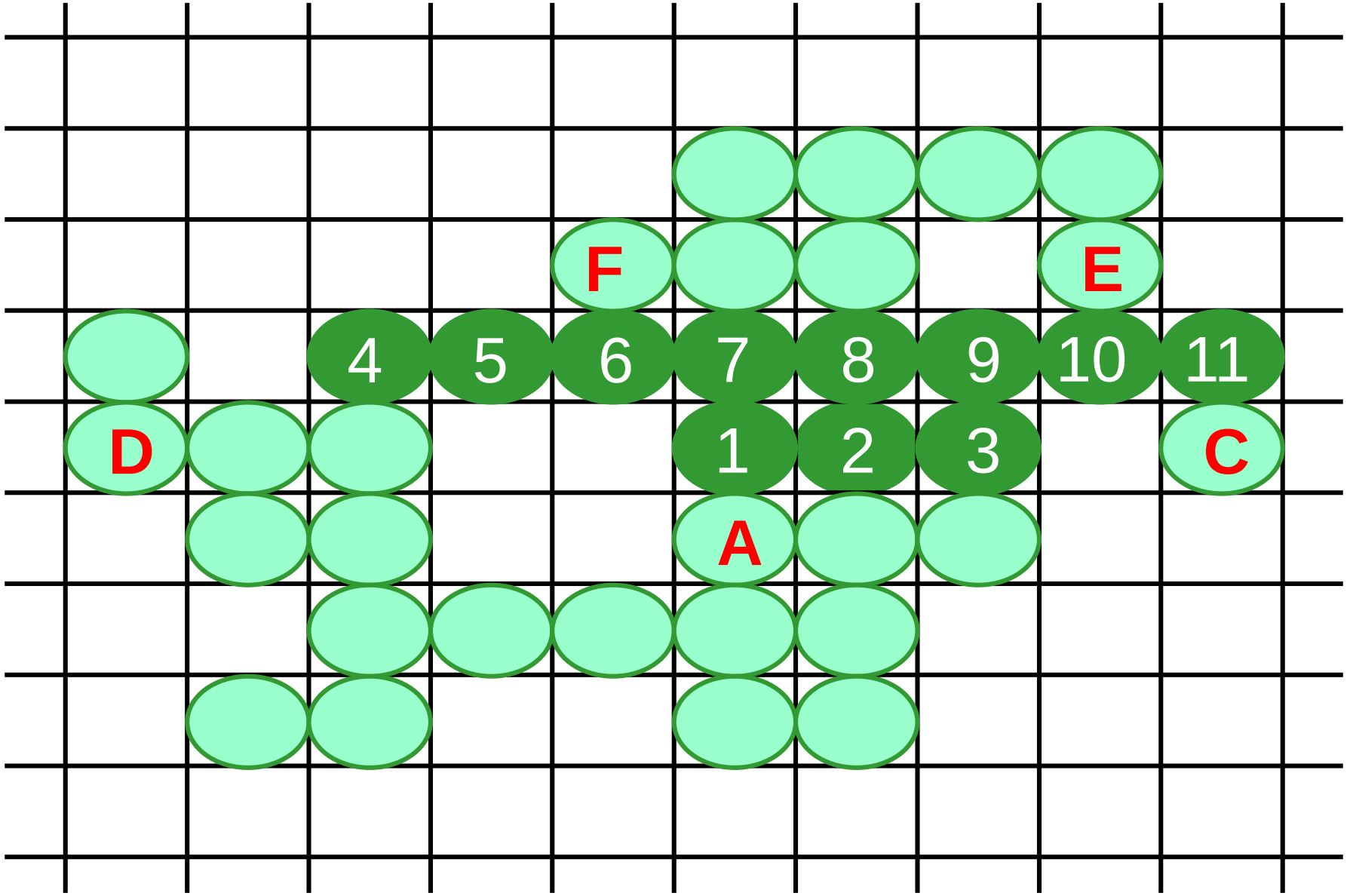
Stack:

x

A

B

Span Flood-Fill Example



Stack:

A

B

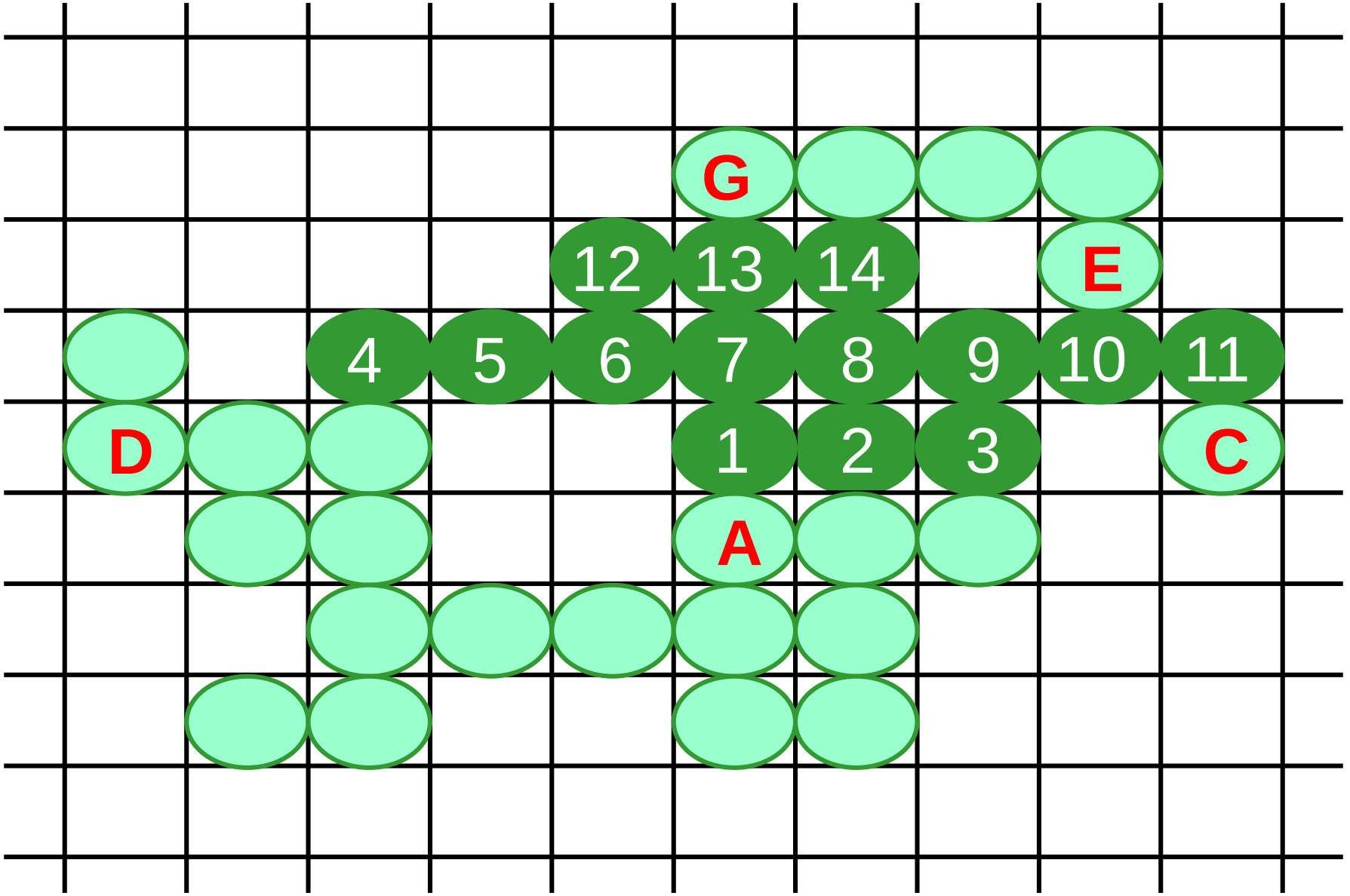
C

D

E

F

Span Flood-Fill Example



Stack:

A

C

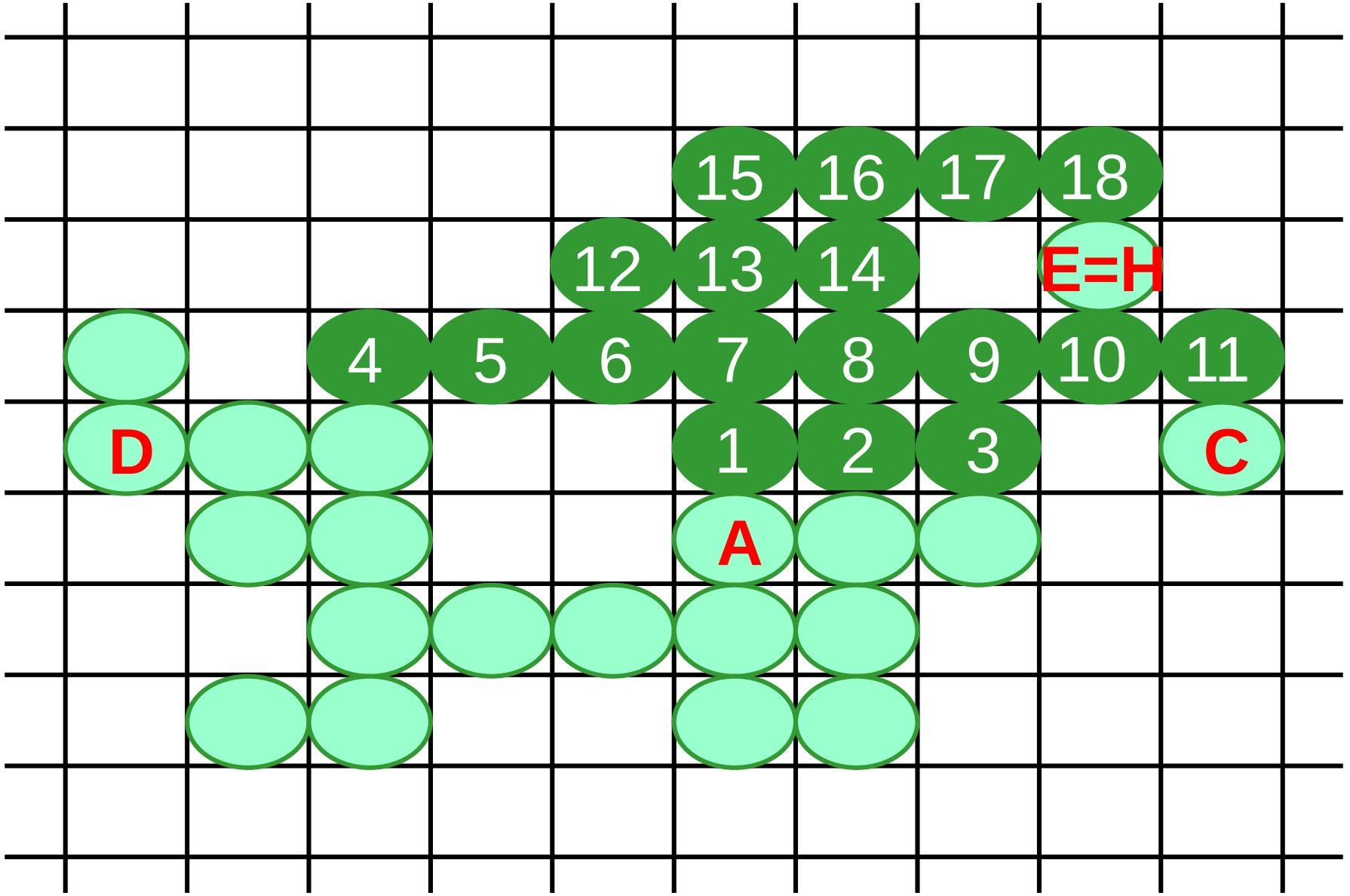
D

E

F

G

Span Flood-Fill Example



Stack:

A

C

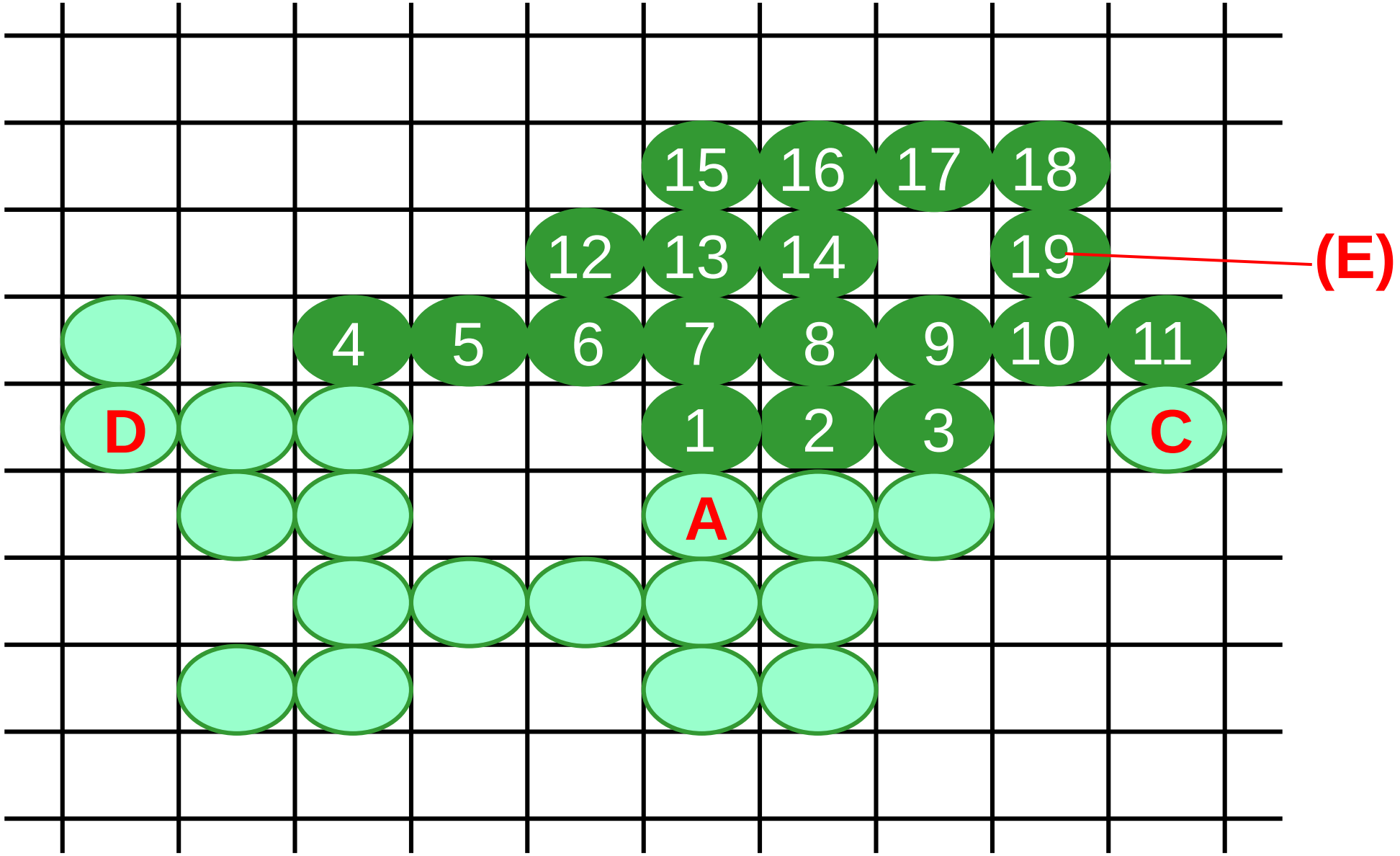
D

E

G

H

Span Flood-Fill Example



Stack:

A

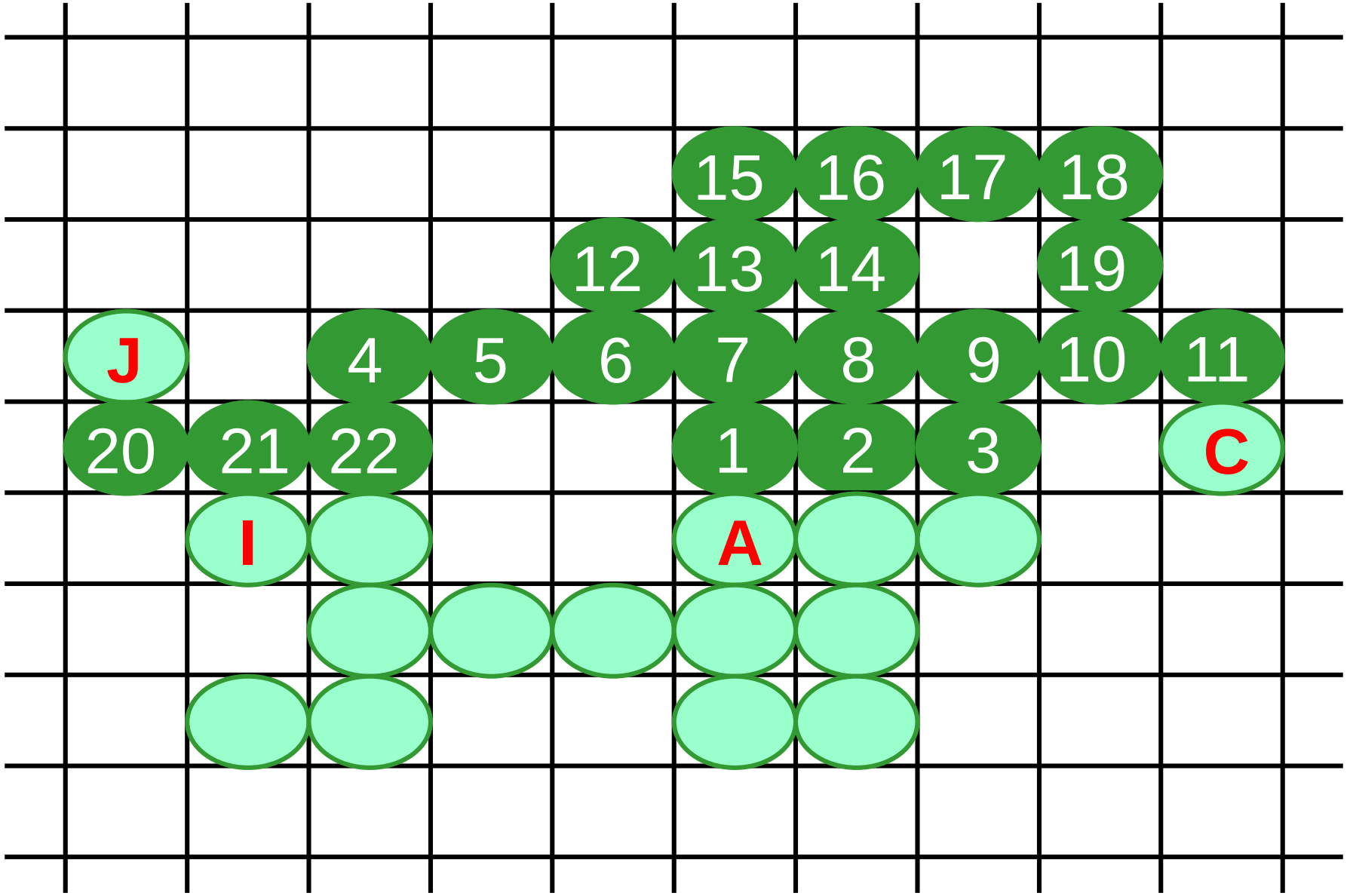
C

D

~~E~~

H

Span Flood-Fill Example



Stack:

A

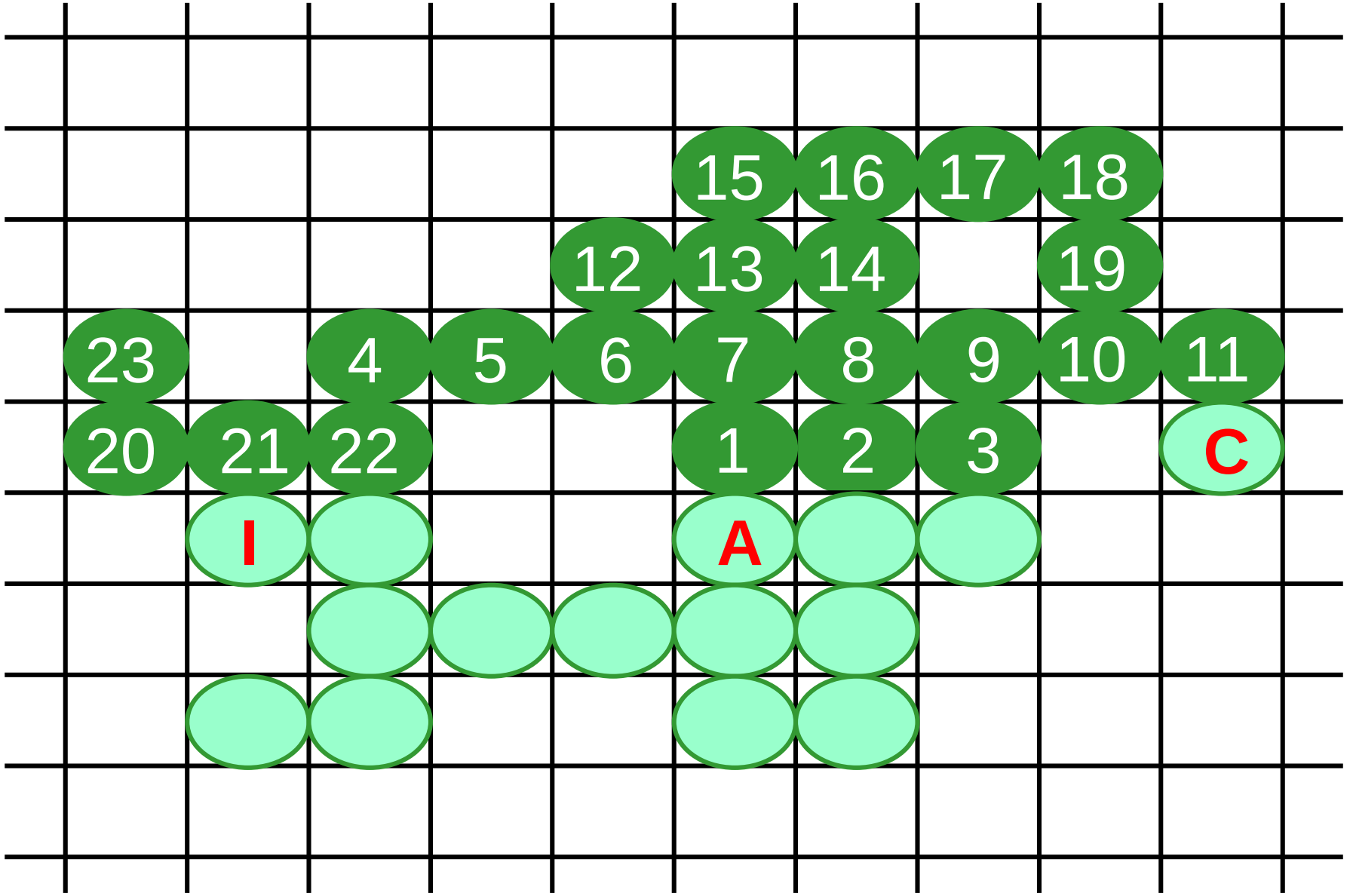
C

D

I

J

Span Flood-Fill Example



Stack:

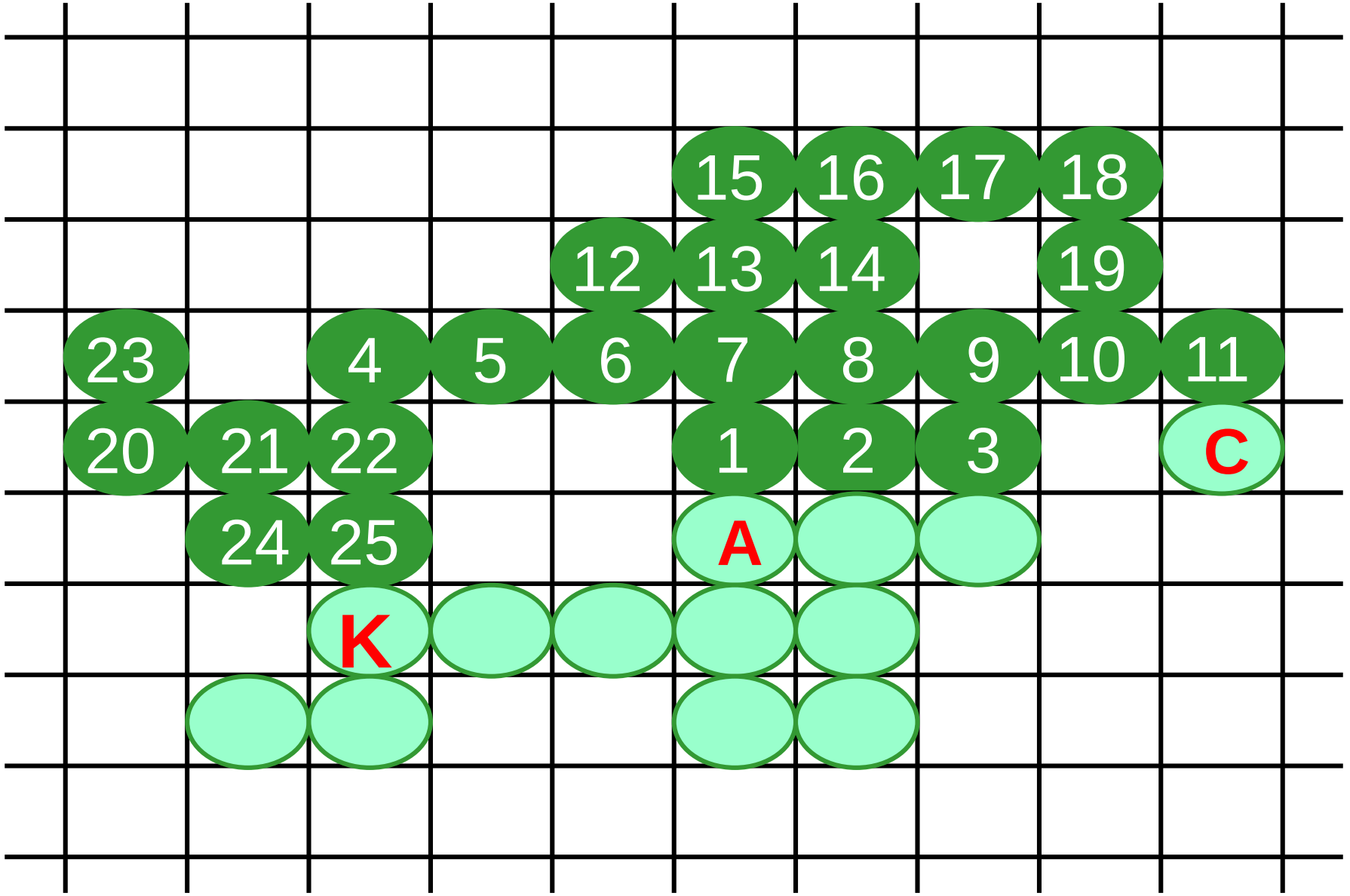
A

C

I

J

Span Flood-Fill Example



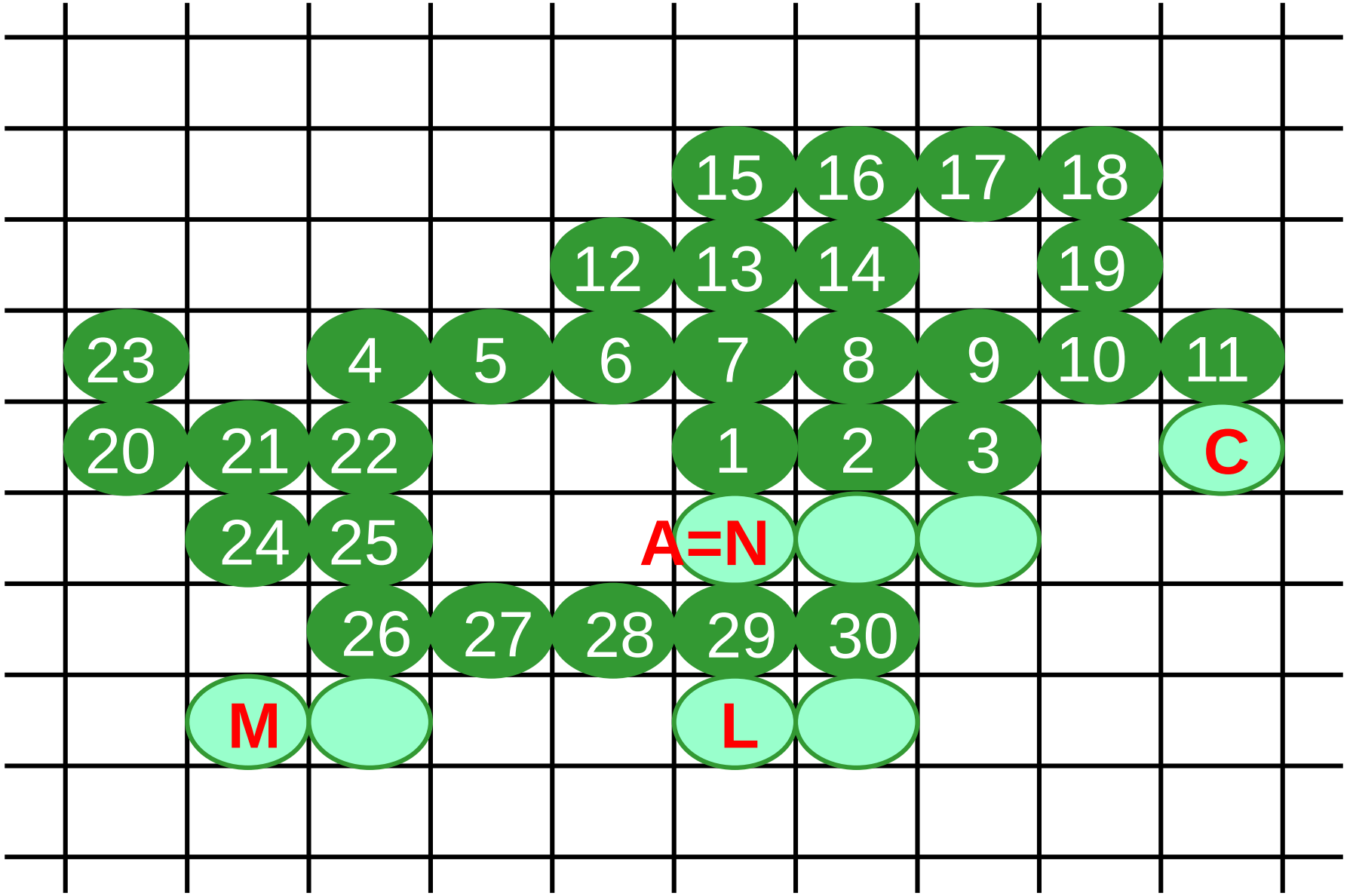
Stack:

A

C

K

Span Flood-Fill Example



Stack:

A

C

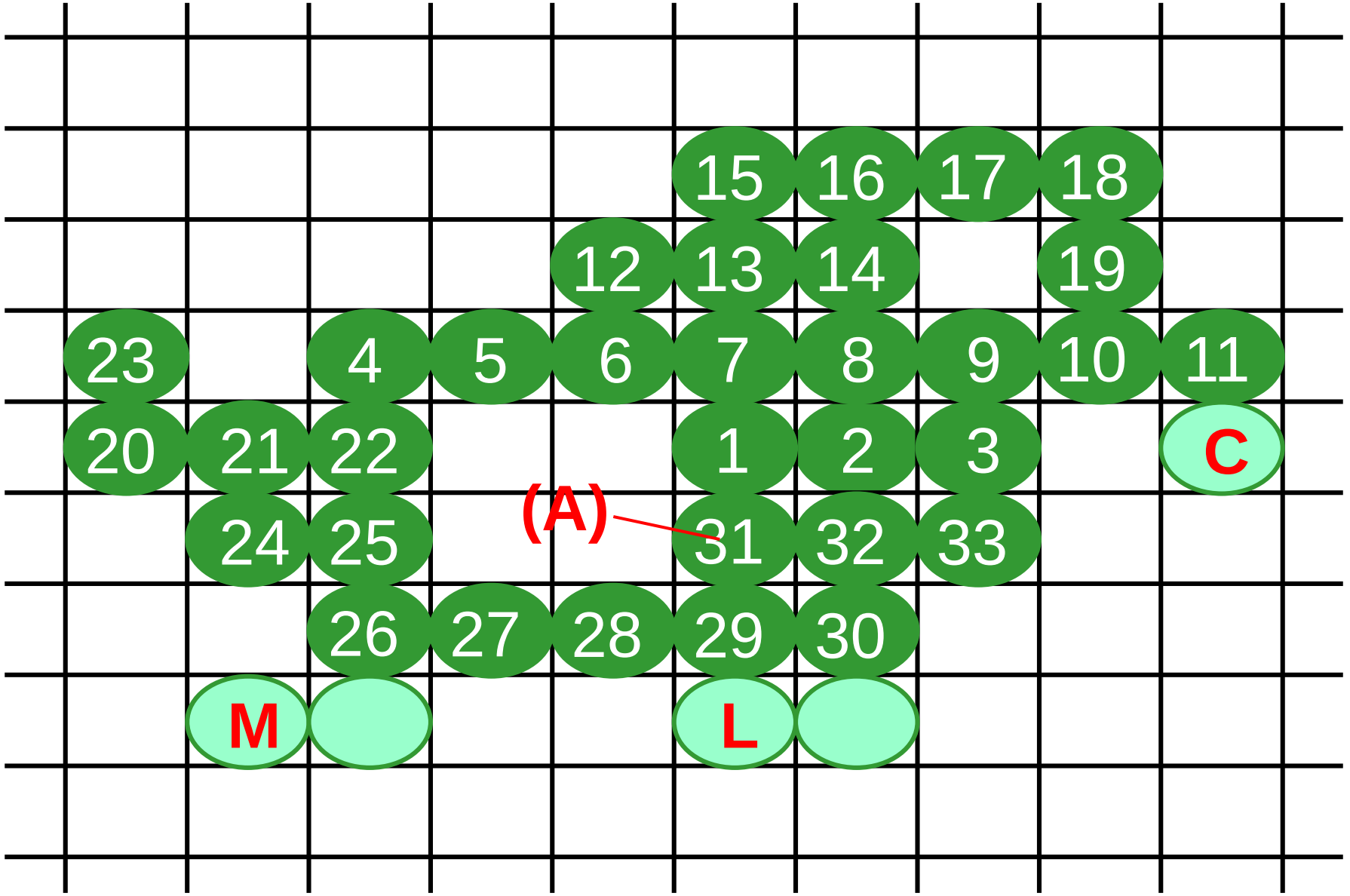
K

L

M

N

Span Flood-Fill Example



Stack:

A

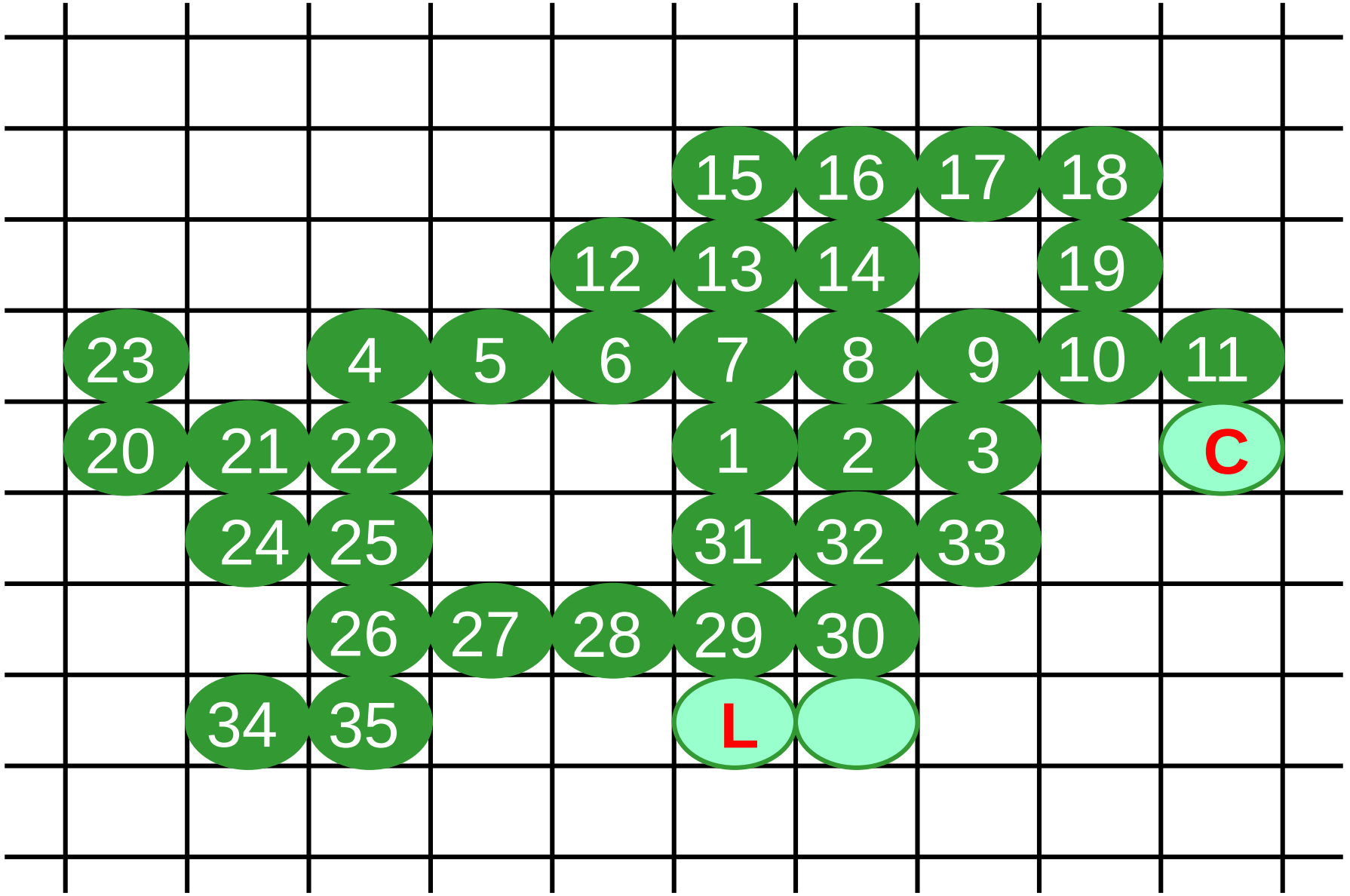
C

L

M

N

Span Flood-Fill Example



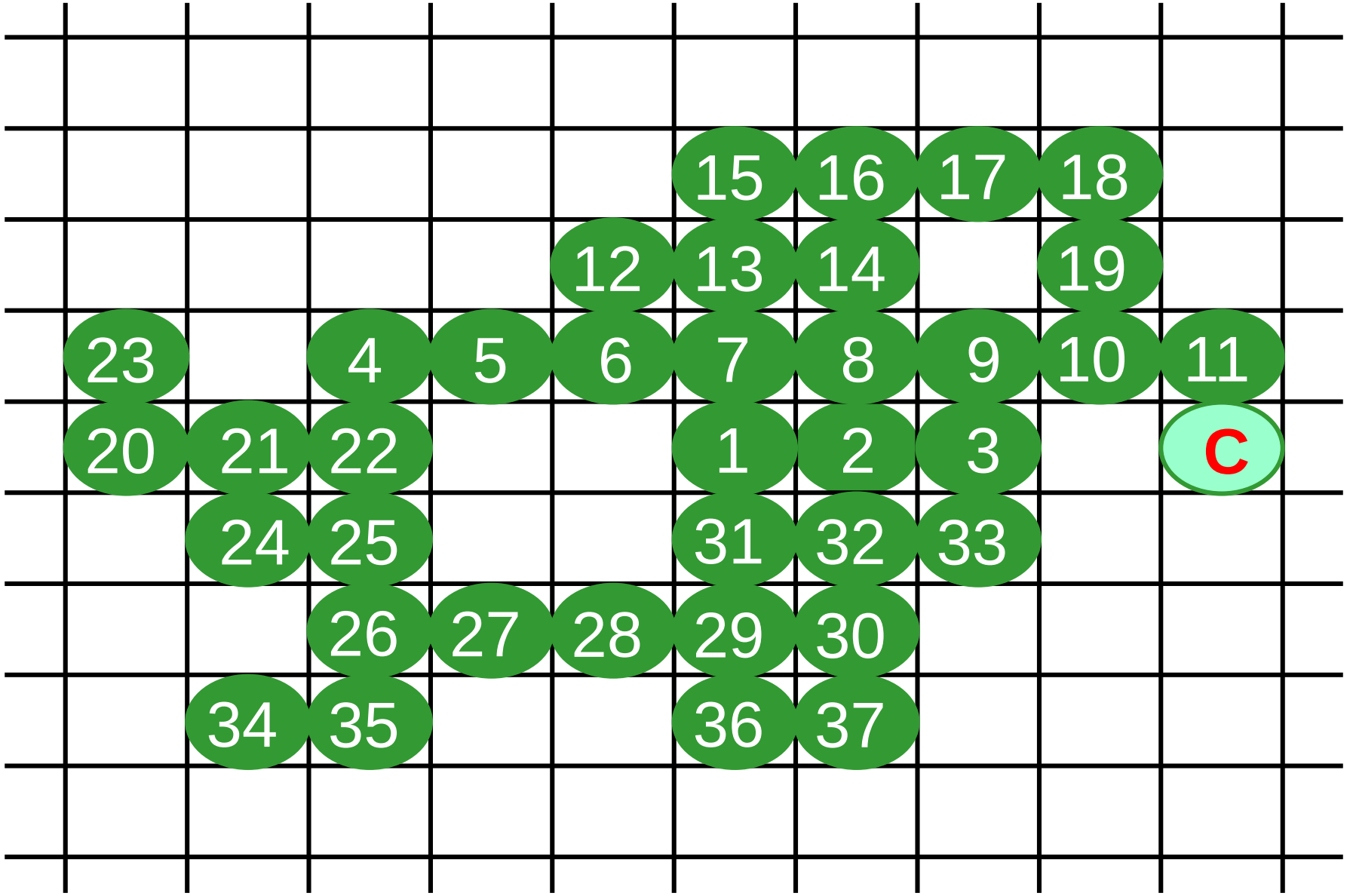
Stack:

C

L

M

Span Flood-Fill Example

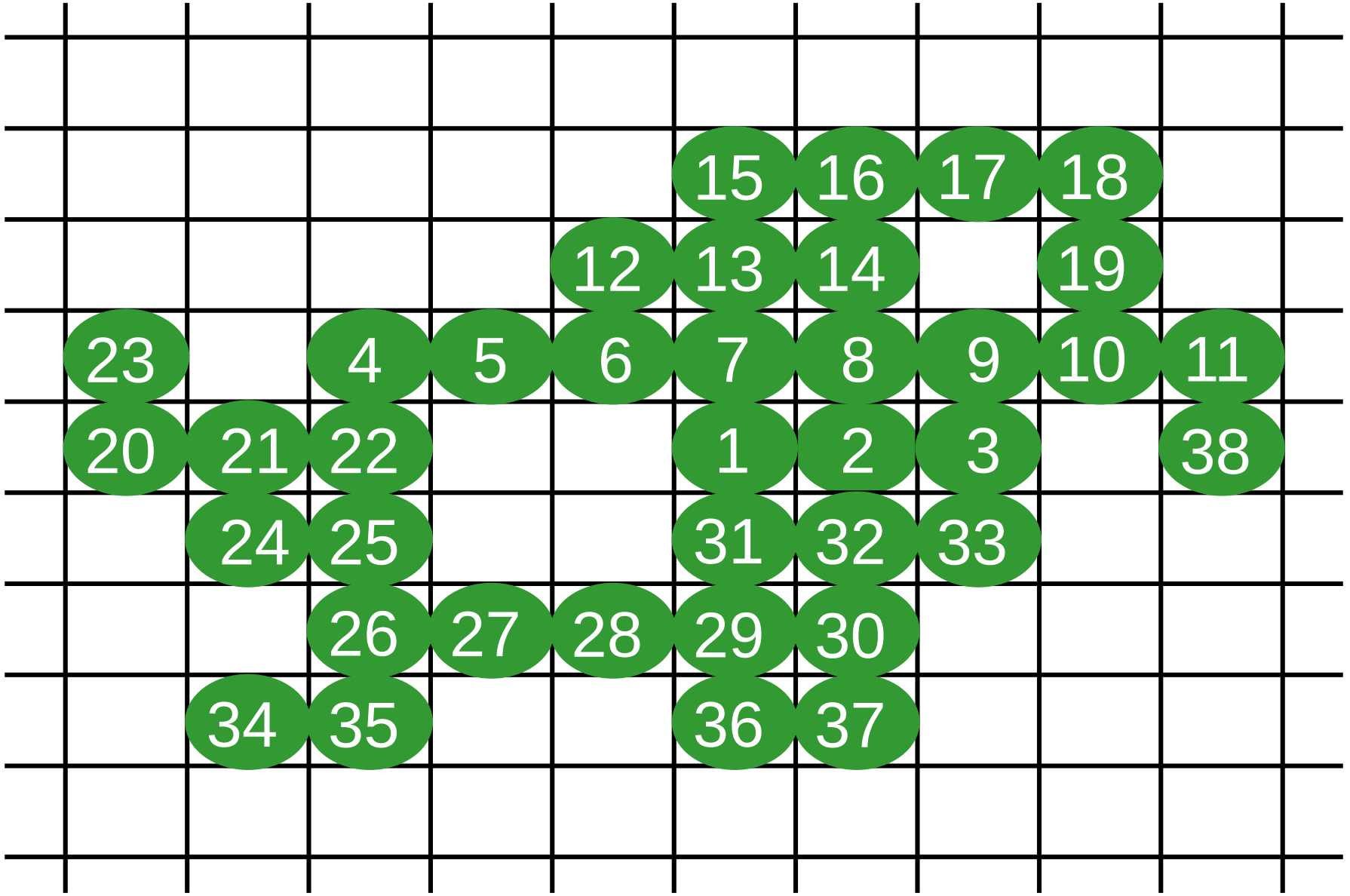


Stack:

C

L

Span Flood-Fill Example

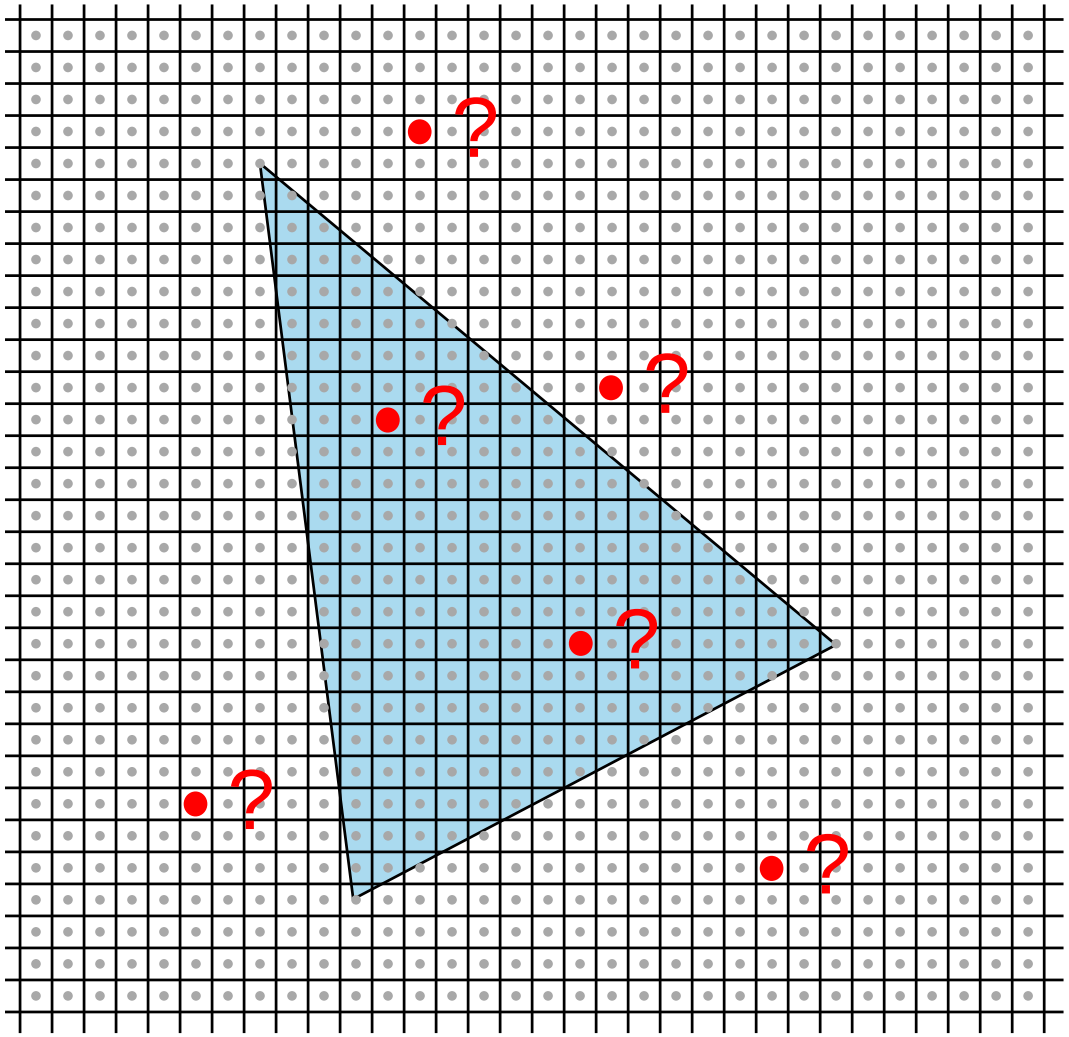


Stack:

C

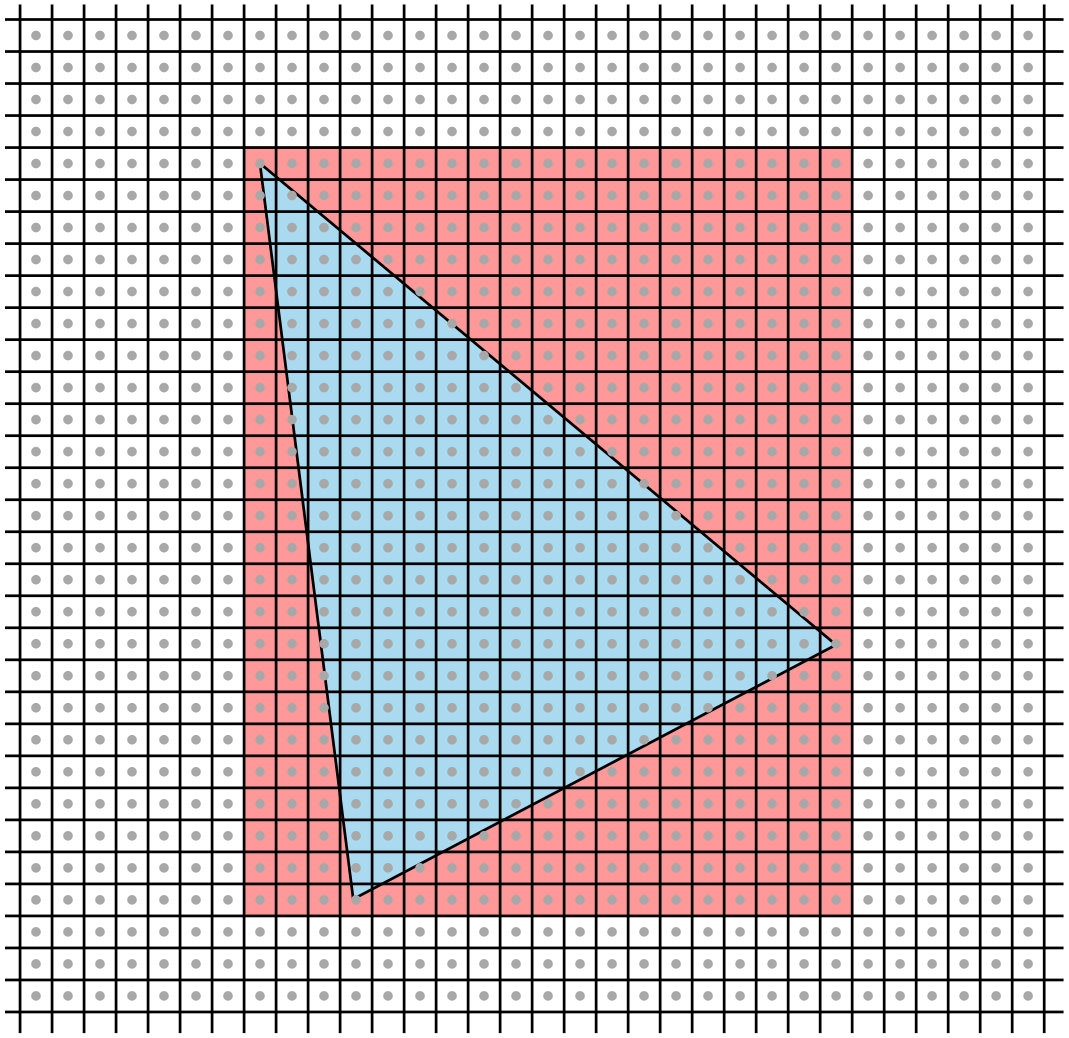
finished!

Parallel Filling of Convex Polygons (e.g. Triangles)



principle:
find out for every pixel separately
if it is inside the polygon !

Parallel Filling of Convex Polygons (e.g. Triangles)

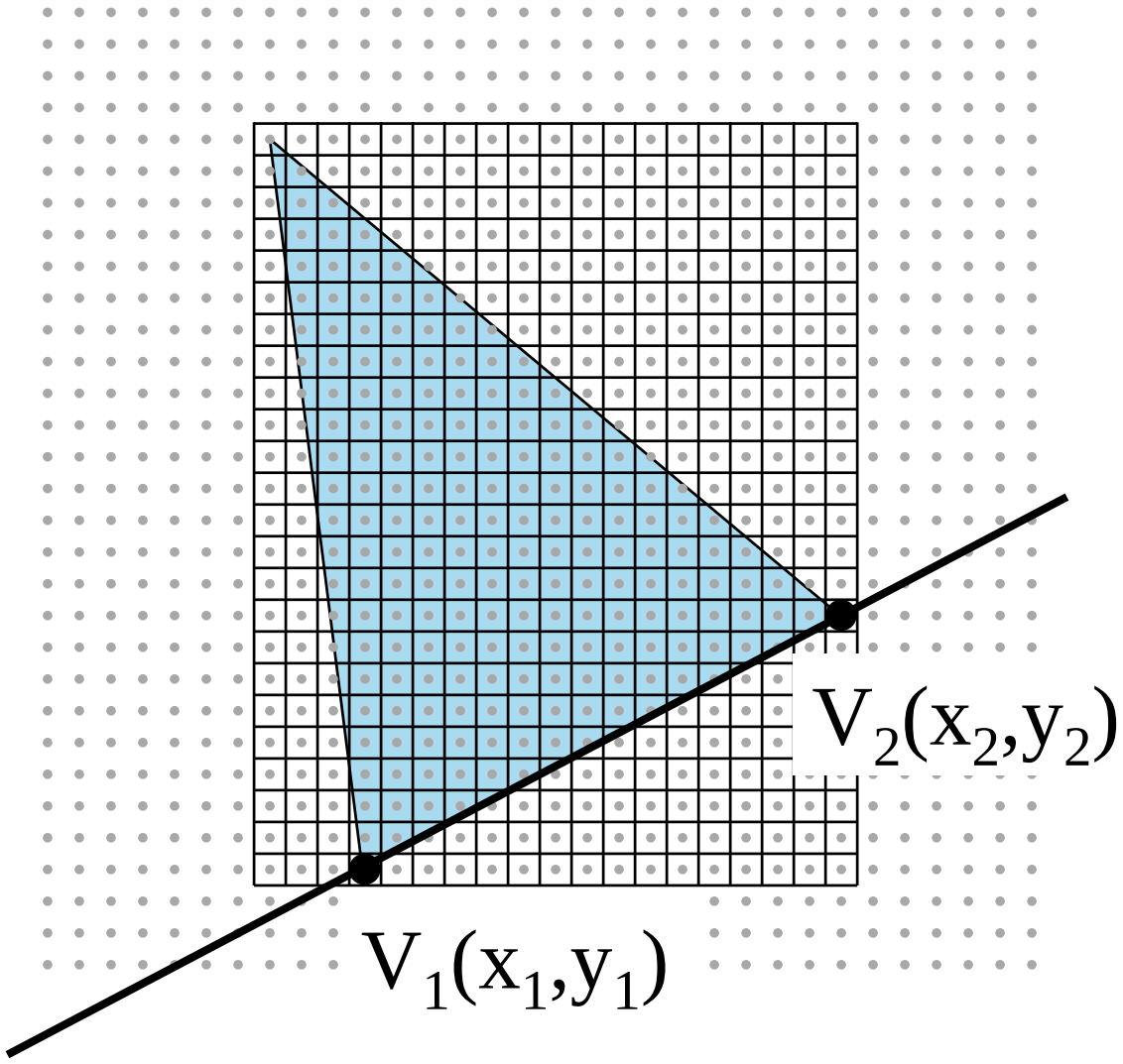


step 1:

restrict to **bounding box**

of the polygon

Parallel Filling of Convex Polygons (e.g. Triangles)

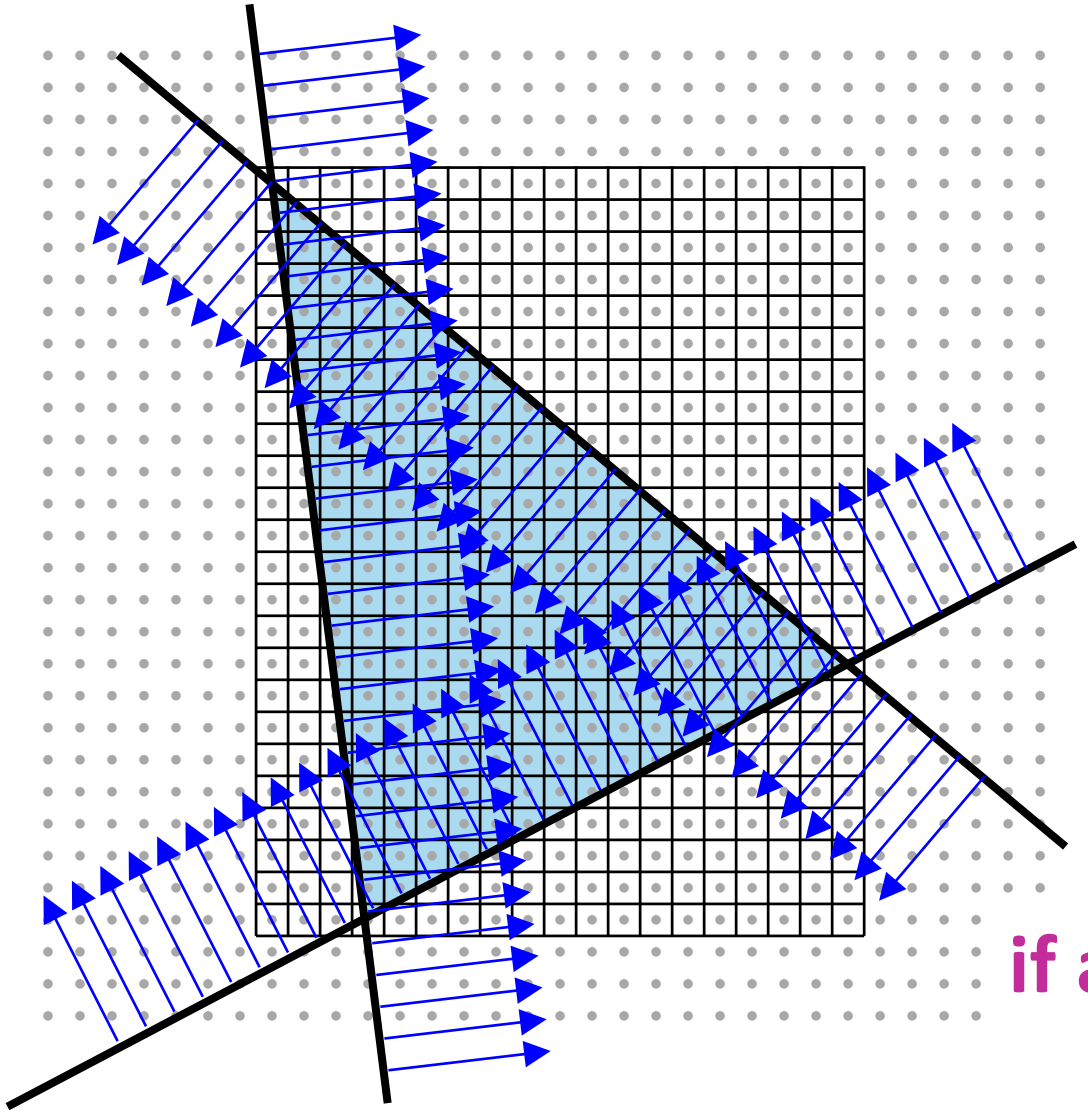


step 2:
calculate **line equations**
of the polygon edges

$$(y_1-y_2)x + (x_2-x_1)y + x_1y_2-x_2y_1 = 0$$

$$Ax + By + C = 0$$

Parallel Filling of Convex Polygons (e.g. Triangles)



step 3:

test each pixel (in parallel !)
against all polygon edges

$Ax + By + C = 0 \Rightarrow (x,y)$ on the line
$Ax + By + C > 0 \Rightarrow (x,y)$ inside
$Ax + By + C < 0 \Rightarrow (x,y)$ outside

if all tests inside ⇒ pixel inside polygon

3D Polygons

- normally surface elements of 3D objects
- attributes are surface attributes
  - color, material, transparency, texture,
    geometric microstructure, reflection properties, …
- attributes and normal vectors often at the vertices
- triangles: linear interpolation with barycentric coordinates